

**AD-A197 242**

7

RADC-TR-88-44  
Final Technical Report  
April 1988

FILE COPY

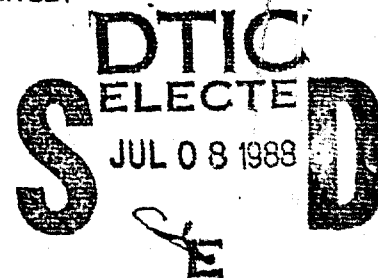


# **EUCLIDEAN DECODERS FOR BCH CODES**

The MITRE Corporation

Willard L. Eastman

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.



ROME AIR DEVELOPMENT CENTER  
Air Force Systems Command  
Griffiss Air Force Base, NY 13441-5700

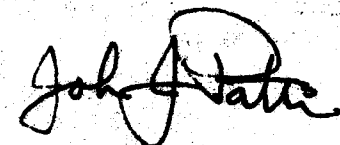
**BEST  
AVAILABLE COPY**

**88 7 07 033**

This report has been reviewed by the RADC Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

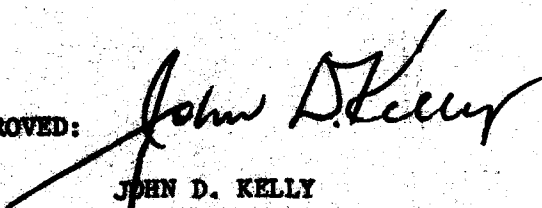
RADC-TR-55-44 has been reviewed and is approved for publication.

APPROVED:



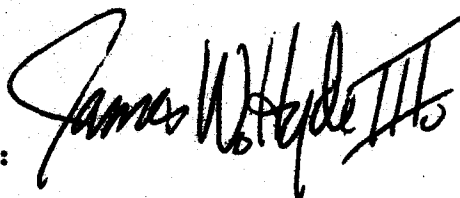
JOHN J. PATTI  
Project Engineer

APPROVED:



JOHN D. KELLY  
Acting Technical Director  
Directorate of Communications

FOR THE COMMANDER:



JAMES W. HYDE, III  
Directorate of Plans & Programs

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (DCCD) Griffiss AFB NY 13441-5700. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

ADA 197242

REPORT DOCUMENTATION PAGE				Form Approved OMB No. 0704-0188	
1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED			1b. RESTRICTIVE MARKINGS N/A		
2a. SECURITY CLASSIFICATION AUTHORITY N/A			3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited		
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A					
4. PERFORMING ORGANIZATION REPORT NUMBER(S) MTR10197			5. MONITORING ORGANIZATION REPORT NUMBER(S) RADC-TR-88-44		
6a. NAME OF PERFORMING ORGANIZATION The MITRE Corporation		6b. OFFICE SYMBOL (if applicable) D82	7a. NAME OF MONITORING ORGANIZATION Rome Air Development Center (DCCD)		
6c. ADDRESS (City, State, and ZIP Code) D82-L-22 Burlington Road Bedford MA 01730			7b. ADDRESS (City, State, and ZIP Code) Griffiss AFB NY 13441-5700		
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Rome Air Development Center		8b. OFFICE SYMBOL (if applicable) DCCD	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F19628-86-C-0001		
8c. ADDRESS (City, State, and ZIP Code) Griffiss AFB NY 13441-5700			10. SOURCE OF FUNDING NUMBERS		
			PROGRAM ELEMENT NO.	PROJECT NO. MOIE	TASK NO. 75
			WORK UNIT ACC'SSION NO. 60		
11. TITLE (Include Security Classification) EUCLIDEAN DECODERS FOR BCH CODES					
12. PERSONAL AUTHOR(S) Willard L. Eastman					
13a. TYPE OF REPORT Final		13b. TIME COVERED FROM Oct 85 TO Oct 86		14. DATE OF REPORT (Year, Month, Day) April 1988	
				15. PAGE COUNT 194	
16. SUPPLEMENTARY NOTATION N/A					
17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)		
FIELD	GROUP	SUB-GROUP			
25	02		Communications		
25	05		Coding		
			Error Correction Codes (615)		
19. ABSTRACT (Continue on reverse if necessary and identify by block number)					
<p>This report investigates conventional decoding algorithms for BCH codes. The algorithm of Sugiyama, Kasahara, Hirasawa and Namekawa, Mills' continued fraction algorithm, and the Berlekamp-Massey algorithm are all viewed as slightly differing variants of Euclid's algorithm. An improved version of Euclid's algorithm for polynomials is developed. The Berlekamp-Massey algorithm is extended within the Euclidean framework to avoid computation of vector inner products. Inversionless forms of the algorithms are considered and the results are extended to provide for decoding of erasures as well as errors.</p> <p>Erasure Chaudhuri Hocquenghem code; Very Large Scale Integration, Tail Decoding of Convolutional Codes</p>					
20. DISTRIBUTION/AVAILABILITY OF ABSTRACT <input checked="" type="checkbox"/> UNCLASSIFIED/UNLIMITED <input type="checkbox"/> SAME AS RPT. <input type="checkbox"/> DTIC USERS			21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED		
22a. NAME OF RESPONSIBLE INDIVIDUAL John J. Patti			22b. TELEPHONE (Include Area Code) (315) 330-3224		22c. OFFICE SYMBOL RADC (DCCD)

DD Form 1473, JUN 86

Previous editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE

UNCLASSIFIED

## EXECUTIVE SUMMARY

This report examines and evaluates three leading conventional decoding algorithms for BCH and Reed-Solomon error-correcting codes:

- the decoding algorithm of Sugiyama et al., which is based on Euclid's algorithm
- a decoding algorithm developed by Scholtz and Welch based on Nills' continued fraction expansion
- the Berlekamp-Massey decoding algorithm.

The three algorithms can be viewed as slightly differing variations of Euclid's algorithm for finding the greatest common divisor of two polynomials. All, in appropriate versions, are suitable for VLSI implementation in a two-dimensional array for pipelined decoding of received codeword polynomials distorted by errors and erasures.

### Extension of the classical decoding theory for BCH codes

The classical decoding theory for  $t$ -error-correcting BCH codes as developed by Peterson, Gorenstein and Zierler, Chien, Forney, and Berlekamp is centered about the key equation

$$Q(x) = S(x)\Lambda(x) \pmod{x^{2t}}$$



iii

Accession For	
NTIS GRA&I	<input checked="checked" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By	
Distribution/	
Availability Codes	
Dist	Special
A-1	



relating three important polynomials:

- the known syndrome polynomial  $S(x)$
- the unknown error locator polynomial  $\Lambda(x)$
- the unknown error evaluator polynomial  $\Omega(x)$

The three conventional algorithms under study solve this equation for the unknown polynomials  $\Lambda(x)$  and  $\Omega(x)$  given the known syndrome polynomial  $S(x)$ . The error locations can then be determined by a Chien search for the zeros of  $\Lambda(x)$  and the error magnitudes can be calculated directly by Forney's formula

$$Y_j = - \frac{\Omega(X_j^{-1})}{\Lambda'(X_j^{-1})}$$

where  $Y_j$  is the  $j$ th error magnitude,  $X_j$  is the field element denoting the  $j$ th error location, and  $\Lambda'(x)$  is the formal derivative of the error locator polynomial.

We have rounded out the classical theory by defining a new polynomial  $A(x)$  such that

$$\Lambda(x)S(x) = x^{2t}A(x) + \Omega(x).$$

The polynomial  $A(x)$  contains the same information as the error evaluator polynomial  $\Omega(x)$  in that the syndrome polynomial can be recovered either from the pair  $(\Omega(x), \Lambda(x))$  or from the pair

$(A(x), \Lambda(x))$ . This leads to the derivation of a new formula for calculation of the error magnitudes in terms of  $A(x)$  and  $\Lambda(x)$ . This new formula (an alternative to Forney's formula) can be used if  $A(x)$  is easier to calculate than  $Q(x)$ .

#### Inside Euclid's algorithm

Euclid's famous algorithm for finding the greatest common divisor of two integers can be immediately generalized for finding the greatest common divisor of two polynomials  $f(x)$  and  $g(x)$  over a given field. In the extended version, the algorithm also yields polynomials  $a(x)$  and  $b(x)$  satisfying

$$\gcd(f(x), g(x)) = a(x)f(x) + b(x)g(x).$$

This form of the algorithm, with suitable modifications, can be used to solve the key equation to produce the error locator polynomial  $\Lambda(x)$  and the error evaluator polynomial  $Q(x)$  (or scalar multiples  $\gamma\Lambda(x)$  and  $\gamma Q(x)$  for some field element  $\gamma$ ), given the syndrome polynomial  $S(x)$ . Euclid's algorithm is the basis both for the decoding algorithm of Sugiyama, et. al. and for the decoding algorithm based on Mills' continued fraction expansion.

Imbedded within Euclid's algorithm is a polynomial division, itself an iterative process, which must be performed once during each iteration of the algorithm. To implement the algorithm in a systolic array, it is desirable to break the polynomial division down into its component sequence of partial divisions, where each partial division consists of a field element inversion, a multiplication of a polynomial by a scalar, and a polynomial subtraction.

We have looked inside Euclid's algorithm to examine the implications of this replacement. When the polynomial divisions are replaced by a sequence of partial divisions, Euclid's algorithm exhibits a two-loop structure; one loop is executed when the partial division does not complete a polynomial division, and the other loop is executed whenever the partial division does complete the polynomial division. (Both loops contain common steps.) A valid, cleaner, and more efficient algorithm can be obtained by deleting one of the loops, with suitable modifications to the remaining loop. The resulting improved algorithm bears a striking resemblance to Berlekamp's algorithm. In effect, this study shows why the Berlekamp-Massey decoding algorithm is more efficient than the decoding algorithms based directly on Euclid's algorithm.

#### The Berlekamp-Massey algorithm in a Euclidean context

Both the Berlekamp-Massey algorithm and the decoding algorithms based upon Euclid's algorithm can be improved by adopting features from each other. The chief drawback of the Berlekamp-Massey algorithm when implemented in a systolic array is the need to calculate a discrepancy between the value of the next syndrome symbol and the next symbol output by the current linear feedback shift register (in Massey's formulation). This calculation requires an inner-product computation at each iteration of the algorithm, a computation whose length increases with the number of iterations.

We have expanded the Berlekamp-Massey algorithm, employing additional polynomials including a remainder-like polynomial  $r(x)$  that corresponds to the remainder polynomial retained in the Euclidean decoding algorithms. Retention of  $r(x)$  obviates the need to calculate the discrepancy at each iteration, for at iteration  $j$  the  $j$ th discrepancy is given by the coefficient  $r_j$ . Thus, at the

cost of additional multiplications and storage, expansion of the Berlekamp-Massey algorithm in a Euclidean context renders the algorithm suitable for VLSI implementation in a two-dimensional systolic array.

#### The Mills' algorithm in a Berlekamp-Massey context

Similarly, the decoding algorithms based on Euclid's algorithm can be improved by modifications that move them closer to Berlekamp's algorithm. The polynomial divisions in Mills' decoding algorithm are replaced by a sequence of partial divisions, again resulting in a two-loop structure. One loop is then removed, yielding an improved decoding algorithm based on our enhanced version of Euclid's algorithm. We have confirmed the validity of the new algorithm by demonstrating that the partial results generated by the algorithm can be mapped by scalar multiplication into partial results generated by its predecessor. The new version of the Mills' algorithm closely resembles the Euclideanized version of the Berlekamp-Massey algorithm, and scalar multiples of the partial results obtained from the one are equated to the partial results obtained from the other. This demonstrates an equivalence among all three of the decoding algorithms studied.

#### Inversionless decoding algorithms

The three decoding algorithms under study and the variants and hybrid versions constructed therefrom all require finite field divisions or, equivalently, inversion of finite field elements. These requirements can be removed, at the cost of further scalar multiplications, by Burton's technique. When Burton's transformation is applied to the Euclideanized Berlekamp-Massey

algorithm and to the enhanced version of the Mills algorithm, the resulting algorithms are identical, except for an algebraic sign in one step.

### Decoding with erasures

Forney has shown that, by defining a (known) erasure locator polynomial  $\kappa(x)$  analogous to the error locator polynomial  $\Lambda(x)$  and defining a modified syndrome polynomial  $T(x)$  by

$$T(x) = \kappa(x)S(x) \quad (\text{mod } x^{2t})$$

one can solve the key equation for errors-and-erasures decoding of  $t$ -error-correcting BCH codes

$$\Omega(x) = \Lambda(x)T(x) \quad (\text{mod } x^{2t})$$

$$= \Pi(x)S(x) \quad (\text{mod } x^{2t})$$

for the errata evaluator polynomial  $\Omega(x)$  and the error locator polynomial  $\Lambda(x)$ . An erratum is either an error or an erasure. The errata locator polynomial  $\Pi(x)$  can then be obtained as

$$\Pi(x) = \kappa(x) \Lambda(x).$$

Forney's formula for calculating the  $j$ th erratum magnitude is rewritten as

$$Y_j = - \frac{\Omega(X_j^{-1})}{\Pi'(X_j^{-1})}$$

where  $\Pi'(x)$  is the formal derivative of  $\Pi(x)$  and  $X_j$  is the field element associated with the  $j$ th erratum location. Substitution of  $T(x)$  for  $S(x)$  in any of the decoding algorithms yields the appropriate values for  $\Lambda(x)$  and  $\Omega(x)$ .

Blahut has shown, however, that the errata locator polynomial  $\Pi(x)$  can be obtained directly from Berlekamp's algorithm if the feedback connection polynomial for the shift-register (in Massey's formulation) is initialized by the erasure locator polynomial  $\kappa(x)$  in place of the polynomial 1. By combining these results, we derive decoding algorithms of the Berlekamp type and of the Euclidean type that yield both the errata locator polynomial  $\Pi(x)$  and the errata evaluator polynomial  $\Omega(x)$ . These algorithms obviate the usual need to calculate  $\Pi(x)$  by a polynomial multiplication of  $\kappa(x)$  and  $\Lambda(x)$ .

### Conclusions

This study has demonstrated that versions of these three BCH decoding algorithms can be constructed that

- allow for the decoding of both errors and erasures
- do not require finite field inversions or divisions
- are suitable for VLSI implementation in a two-dimensional systolic array, allowing pipelining of the received codeword polynomials.

This implementation will be the subject of further study.

## ACKNOWLEDGMENTS

This study was performed under MOIE Project 7560, Error Control Coding and Modulation Techniques, funded by the Rome Air Development Center, U.S. Air Force Electronic Systems Division, Griffiss Air Force Base, NY, under Contract No. F19628-86-C-0001.

The author has received an unusual level of assistance from numerous colleagues in Department D-82, including J.G. Bressel, J.H. Cozzens, R.A. Games, B.L. Johnson, S.J. Meehan, D.J. Muder, and J.J. Vaccaro. In particular, Dr. John Cozzens read and criticized the entire manuscript, offering many helpful suggestions for its improvement, and Project Leader Bruce Johnson devoted many hours to careful study of earlier drafts, provided many fruitful suggestions, and helped with an extensive reshaping of all parts of the report. T.J. McDonald provided editorial assistance, R.C. McLeman and H.K. Conroy typed the manuscript. For all this support the author is sincerely grateful.

## TABLE OF CONTENTS

<u>Section</u>	<u>Page</u>
1 INTRODUCTION	1
1.1 PURPOSE	1
1.2 BACKGROUND	2
1.3 SCOPE	3
2 EUCLID'S ALGORITHM	7
3 THE DECODING PROBLEM	19
3.1 THE CLASSICAL BCH DECODING THEORY	19
3.2 EXTENSION OF THE CLASSICAL THEORY	28
4 THE JAPANESE DECODING ALGORITHM	43
5 MILLS' CONTINUED FRACTIONS ALGORITHM	53
5.1 CONTINUED FRACTIONS AND EUCLID'S ALGORITHM	53
5.2 DECODING BCH CODES BY MILLS' ALGORITHM	64
6 THE BERLEKAMP-MASSEY ALGORITHM	71
7 HYBRIDS AND COMPARISONS	83
7.1 THE BERLEKAMP-MASSEY ALGORITHM IN EUCLIDEAN DRESS	83
7.2 CITRON'S ALGORITHM	92
7.3 INSIDE EUCLID'S ALGORITHM	105
7.4 MILLS' ALGORITHM IN BERLEKAMP-MASSEY DRESS	118
7.5 COMPARISONS	137



## TABLE OF CONTENTS (Concluded)

<u>Section</u>		<u>Page</u>
8	INVERSIONLESS DECODING	143
	8.1 BURTON'S ALGORITHM	143
	8.2 INVERSIONLESS EUCLIDEAN ALGORITHMS	148
9	DECODING ERASURES	155
10	CONCLUSION	171
REFERENCES		173

## LIST OF ILLUSTRATIONS

<u>Figure</u>		<u>Page</u>
1	An L-Stage Linear Feedback Shift Register	72
2	Construction of $c^{(n+1)}(x) = c^{(n)}(x) - d_n d_m^{-1} x^{n-m} c^{(m)}(x)$	77

## LIST OF TABLES

<u>Table</u>		<u>Page</u>
1	A Comparison of Outputs from Programs 12 ( $r^N(x)$ ) and 8 ( $R^N(x)$ )	131
2	A Comparison of Outputs from Programs 9 ( $r^T(x)$ ) and 10 ( $R^T(x)$ )	136
3	Number of Multiplications Required for Obtaining $\Lambda(x)$ in the Presence of $t$ Errors	142

# LIST OF PROGRAMS

<u>Program</u>		<u>Page</u>
1	Euclid's Algorithm	9
2	Extended Euclid's Algorithm	12
3	Euclid's Algorithm for Polynomials Over $GF(q)$	15
4	Japanese Decoding Algorithm	49
5	Mills' Decoding Algorithm	67
6	Berlekamp-Massey Algorithm	80
7	Euclideanized Berlekamp-Massey Algorithm	88
8	Citron's Algorithm	102
9	Euclid's Algorithm Without Polynomial Division	108
10	Simplified Euclid's Algorithm	115
11	Mills' Algorithm with Partial Divisions	119
12	Simplified Mills' Algorithm	126
13	Burton's Algorithm	144
14	Inversionless Mills' Algorithm	149
15	Burtonized Mills' Algorithm - Final Version	151
16	Decoding with Erasures	162
17	Decoding with Erasures - Japanese Algorithm	169

## SECTION 1

### INTRODUCTION

#### 1.1 PURPOSE

The objective of Project 7560, Error Control Codes and Modulation Techniques, is to identify and develop decoding algorithms that lead to architectures well suited for very large scale integrated (VLSI) circuit technology implementation. The early thrust of the FY86 activities was to develop an awareness of current research in convolutional coding, signal-space coding, and algebraic block coding. We concluded that algebraic block coding held the most unfulfilled potential for VLSI implementation, and then focused project activities in this area.

Three leading algorithms for the decoding of Bose-Chaudhuri-Hocquenghem (BCH) codes and Reed-Solomon codes were selected for study and comparison. These were the decoding algorithm of Sugiyama, Kasahara, Hirasawa, and Namekawa; Mills' continued-fraction algorithm; and the Berlekamp-Massey algorithm. Hybrid versions of the algorithms were developed, and enhancements that improve the efficiency of the algorithms or their suitability for VLSI implementation were proposed. This report documents the results of our investigation.

The report is intended to provide the circuit designer with a thorough understanding of the decoding algorithms. The approach taken is a descriptive rather than a formal mathematical one. Each algorithm is described concisely and precisely using Iverson's programming language (APL). Each algorithm is illustrated by a decoding example. Formal theorems and proofs are avoided throughout

the report, but the attempt is made to show how and why each algorithm works. Estimates are made of the complexity (number of scalar multiplications) and throughput (number of cycle times or systolic array levels) associated with each algorithm.

We assume that the reader has some knowledge of the basics of algebraic coding theory. The design and structure of BCH codes are not discussed. Encoding is not discussed. Decoding is discussed in great detail, but with emphasis placed on solution of the so-called key equation. Each of the three algorithms solves the key equation for the error locator and error evaluator polynomials, given the syndrome polynomial derived from the received codeword polynomial. This is the critical section of an algebraic decoder.

## 1.2 BACKGROUND

The advent of BCH codes [1-3] gave birth to a flurry of activity in the design of algebraic decoders. Peterson [4,5] developed the fundamental algorithm, based upon inversion of finite field matrices, for decoding binary BCH codes. Gorenstein and Zierler [6] extended Peterson's algorithm to nonbinary BCH codes, noting that the codes of Reed and Solomon [7] form a special case of nonbinary BCH codes. Chien [8] proposed a search method for deriving the error locations from the error locator polynomial. Forney [9] gave a direct formula for calculating the error magnitudes from the error evaluator polynomial and the formal derivative of the error locator polynomial and also developed a method for decoding erasures.

Berlekamp [10] developed an efficient algorithm for determining the error locator and error evaluator polynomials. Massey [11]

elucidated Berlekamp's algorithm by deriving it as a method for synthesizing the shortest linear feedback shift register (LFSR) that will generate a given sequence. The algorithm now is frequently called the Berlekamp-Massey algorithm.

Sugiyama, Kasahara, Hirasawa, and Namekawa [12] employed Euclid's algorithm to solve the key equation for the error locator polynomial and the error evaluator polynomial. Mills [13] gave a continued-fraction algorithm for finding linear recurrences. Mills' algorithm is in essence the same as the algorithm of Sugiyama et al. Welch and Scholtz [14] showed an equivalence between Mills' algorithm and the Berlekamp-Massey algorithm.

We regard these last three algorithms as variants of Euclid's algorithm. Viewing the Berlekamp-Massey algorithm in a Euclidean framework, for instance, provides a deeper insight into its workings, leading to computational simplification and to the elicitation of further information from its employment.

### 1.3 SCOPE

This report contains a detailed examination of these three algebraic decoding algorithms proposed for the decoding of BCH error-correcting codes. We compare the suitability of the algorithms for VLSI implementation. All three algorithms are viewed essentially as variants of Euclid's extended algorithm for polynomials. Several versions, including hybrids, of these algorithms are developed and compared. Each version is represented

in the form of a program. This provides both concision and precision in the description of each algorithm, highlighting the similarities and differences between different variants.

Section 2 of the report reviews Euclid's algorithm. Section 3 consists of two parts: a brief review of the decoding problem for BCH codes and of the classical decoding algorithms as developed by Peterson, Gorenstein and Zierler, Chien, and Forney; and an extension of the classical development, providing an alternative to Forney's formula for calculating the error magnitudes.

Sections 4, 5, and 6 contain reviews of the three algebraic decoding algorithms under consideration. Section 4 examines the algorithm of Sugiyama, Kasahara, Hirasawa, and Namekawa. Section 5 explores the relationship between Euclid's algorithm and Mills' continued-fraction expansion. The decoding algorithm obtained by Scholtz and Welch from Mills' algorithm is examined and shown to be essentially the same as the algorithm of Sugiyama et al. Section 6 reviews the decoding algorithm invented by Berlekamp and rederived by Massey in the context of LFSR synthesis.

Section 7 contains the main results of the report and is divided into five parts. In the first part, the Berlekamp-Massey algorithm is expanded in a Euclidean context by the calculation of additional polynomials analogous to those computed in the extended Euclid's algorithm. The resulting algorithm is more efficient for VLSI implementation because the inner-product calculation of

discrepancies is obviated. Section 7.2 examines a new decoding algorithm proposed by Citron, and shows that it belongs to the class of Euclideanized Berlekamp-Massey algorithms developed in section 7.1. Euclid's algorithm for polynomials is dissected in section 7.3. The polynomial divisions inherent in the algorithm are first separated into their component partial divisions. The resulting algorithm is then modified and rearranged into a cleaner, more efficient version of Euclid's algorithm. In section 7.4 this same dissection and modification are applied to Mills' decoding algorithm. The resulting algorithm, incorporating the more efficient version of Euclid's algorithm, is more efficient for decoding and closely parallels the Euclideanized versions of the Berlekamp-Massey algorithm. An equivalence between Mills' algorithm and the Berlekamp-Massey algorithm is then established by demonstrating that the partial results obtained by the various versions can be mapped into one another by multiplication by suitable scalar factors. Section 7.5 summarizes the similarities and differences among the several algorithms and their variants.

In section 8, the decoding algorithms are modified, using a technique developed by Burton, to avoid all finite field division or inversion. The resulting versions of Mills' algorithm and the Berlekamp-Massey algorithm are seen to be nearly identical. In section 9, results are extended to include the decoding of erasures as well as errors. Results of Forney and of Blahut can be combined to give decoding algorithms that directly furnish both the errata locator polynomial and the errata evaluator polynomial. Section 10 summarizes the conclusions of the study.



SECTION 2  
EUCLID'S ALGORITHM

In Book VII, Proposition 2 of his Elements [15], Euclid gave his famous algorithm for finding the greatest common divisor  $\gcd(s,t)$  of two integers  $s$  and  $t$ .

Let  $s > t > 0$ . Dividing  $s$  by  $t$ ,

$$s = q_1 t + r_1, \quad (0 \leq r_1 < t)$$

and, if  $r_1 > 0$ , the  $\gcd(s,t)$  must also divide the remainder  $r_1$ . Continuing the process if  $r_1 \neq 0$ ,

$$t = q_2 r_1 + r_2, \quad (0 \leq r_2 < r_1),$$

etc.

$$r_k = q_{k+2} r_{k+1} + r_{k+2}, \quad (0 \leq r_{k+2} < r_{k+1})$$

until finally, for some  $n$ ,  $r_n$  must equal 0:

$$r_{n-2} = q_n r_{n-1} + 0.$$

Since  $r_{n-1}$  divides  $r_{n-2}$ , it must divide  $r_{n-3} = q_{n-1} r_{n-2} + r_{n-1}$ , and, similarly, all  $r_k$ , back to  $r_0 = t$  and  $r_{-1} = s$ . Thus,  $r_{n-1}$  is a common divisor of  $s$  and  $t$ , and since the  $\gcd(s,t)$  must

divide each nonzero remainder  $r_k$ ,  $r_{n-1}$  must be the greatest common divisor of  $s$  and  $t$ . The recursion of Euclid's algorithm is given by

$$r_k = r_{k-2} - q_k r_{k-1}. \quad (1)$$

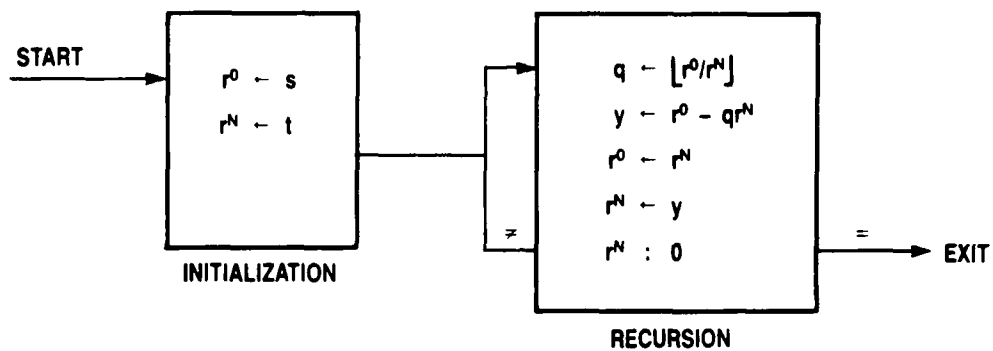
A simple program for Euclid's algorithm is shown in program 1. The programming notation is loosely adapted from Iverson [16]. The back-arrow can be read as "is specified by," the colon as "is compared to." For each specification statement, the quantity to the left of the arrow is replaced by the quantity to the right. For comparison statements, branches leaving the statement at either side are followed if, and only if, the relation represented by the branch label is satisfied when substituted for the colon in the comparison statement. A branch with no label is always followed.

The notation  $[a/b]$  denotes the integer part or quotient of the division of  $a$  by  $b$ .  $r^O$  and  $r^N$  represent the "old" and "new" values of the remainder  $r_k$  at iteration  $k$ . At the beginning of the  $k$ th iteration,  $r^O$  is  $r_{k-2}$  and  $r^N$  is  $r_{k-1}$ ; at the end of the  $k$ th iteration,  $r^O$  is  $r_{k-1}$  and  $r^N$  is  $r_k$ .

Example 1:  $s = 9022$ ,  $t = 4719$ .

Initialization:  $r^O \leftarrow 9022$   
 $r^N \leftarrow 4719$

Iteration 1:  $q \leftarrow 1$   
 $y \leftarrow 4303$   
 $r^O \leftarrow 4719$   
 $r^N \leftarrow 4303$



INPUT: INTEGERS  $s > t > 0$

OUTPUT:  $r^0 = \text{gcd}(s, t)$

Program 1. EUCLID'S ALGORITHM

Iteration 2:       $q \leftarrow 1$   
                   $y \leftarrow 416$   
                   $r_0 \leftarrow 4303$   
                   $r_N \leftarrow 416$

Iteration 3:       $q \leftarrow 10$   
                   $y \leftarrow 143$   
                   $r_0 \leftarrow 416$   
                   $r_N \leftarrow 143$

Iteration 4:       $q \leftarrow 2$   
                   $y \leftarrow 130$   
                   $r_0 \leftarrow 143$   
                   $r_N \leftarrow 130$

Iteration 5:       $q \leftarrow 1$   
                   $y \leftarrow 13$   
                   $r_0 \leftarrow 130$   
                   $r_N \leftarrow 13$

Iteration 6:       $q \leftarrow 10$   
                   $y \leftarrow 0$   
                   $r_0 \leftarrow 13$   
                   $r_N \leftarrow 0$

At termination of the example, when  $r_N = 0$ , the gcd (9022,4719) is found to be 13. Tracing through all the remainders  $r_0$  and  $r_N$  for the six iterations yields

$\text{gcd}(9022,4719) = \text{gcd}(4719,4303)$   
                   $= \text{gcd}(4303,416)$   
                   $= \text{gcd}(416,143)$   
                   $= \text{gcd}(143,130)$   
                   $= \text{gcd}(130,13)$   
                   $= \text{gcd}(13,0)$   
                   $= 13$

Euclid's algorithm also will provide integers  $a$  and  $b$  which satisfy the equation

$$a \cdot s + b \cdot t = \text{gcd}(s,t). \quad (2)$$

To achieve this the initial values  $a_{-1}$ ,  $a_0$ ,  $b_{-1}$ , and  $b_0$  are chosen as (1,0,0,1). A recursion analogous to (1) is then applied to  $a$  and  $b$ :

$$\begin{aligned} a_k &= a_{k-2} - q_k a_{k-1} \\ b_k &= b_{k-2} - q_k b_{k-1} \end{aligned} \quad (3)$$

It follows, by induction on  $k$ , that at each iteration  $k$ ,

$$a_k \cdot s + b_k \cdot t = r_k. \quad (4)$$

Program 2 is a modification of program 1 which provides  $a$  and  $b$ .

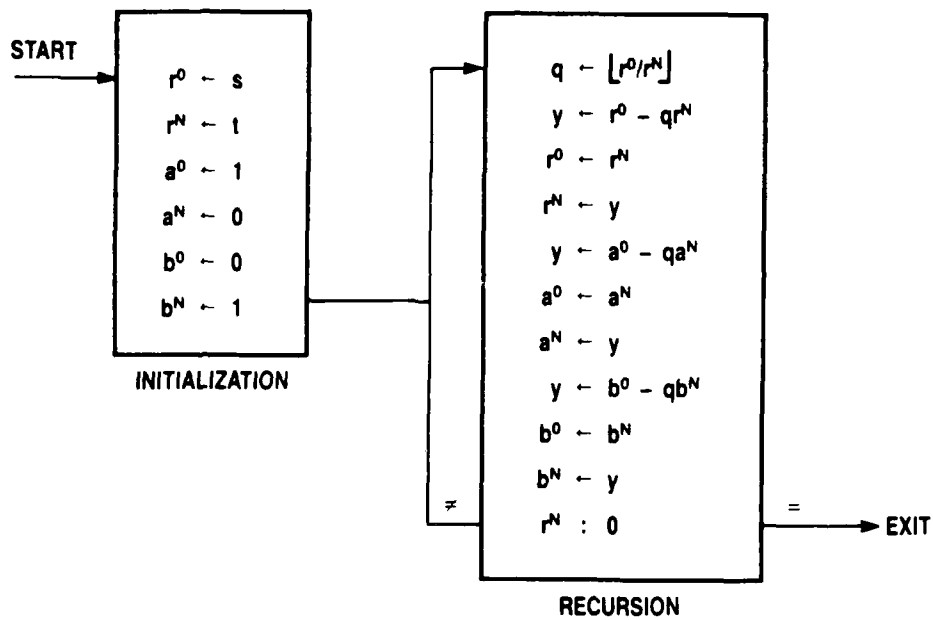
Example 1 (continued):  $s = 9022$ ,  $t = 4719$ .

$k$	$q$	$r^0$	$r^N$	$a^0$	$a^N$	$b^0$	$b^N$
0	-	9022	4719	1	0	0	1
1	1	4719	4303	0	1	1	-1
2	1	4303	416	1	-1	-1	2
3	10	416	143	-1	11	2	-21
4	2	143	130	11	-23	-21	44
5	1	130	13	-23	34	44	-65
6	10	13	0	34	-363	-65	694

At termination

$$34 \cdot 9022 + (-65) \cdot 4719 = 13$$

satisfying (2).



INPUT: INTEGERS  $s > t > 0$

OUTPUT:  $r^0 = \gcd(s, t) = a^0s + b^0t$

**Program 2. EXTENDED EUCLID'S ALGORITHM**

The recursion (3) for  $a_k$  and  $b_k$  can be written in matrix form as

$$\begin{aligned}
 \begin{bmatrix} a_k & a_{k-1} \\ b_k & b_{k-1} \end{bmatrix} &= \begin{bmatrix} a_{k-1} & a_{k-2} \\ b_{k-1} & b_{k-2} \end{bmatrix} \begin{bmatrix} -q_k & 1 \\ 1 & 0 \end{bmatrix} \\
 &= \begin{bmatrix} a_{k-2} & a_{k-3} \\ b_{k-2} & b_{k-3} \end{bmatrix} \begin{bmatrix} -q_{k-1} & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} -q_k & 1 \\ 1 & 0 \end{bmatrix} \\
 &\quad \vdots \\
 &= \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} -q_1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} -q_2 & 1 \\ 1 & 0 \end{bmatrix} \cdots \begin{bmatrix} -q_k & 1 \\ 1 & 0 \end{bmatrix}
 \end{aligned}$$

Taking determinants on both sides,

$$a_k b_{k-1} - b_k a_{k-1} = (-1)^{k+1} \quad (5)$$

which gives the useful result that  $\gcd(a_k, b_k) = 1$  for all  $k$ .

Euclid's algorithm can be immediately extended to polynomials and, in particular, to polynomials over a finite field  $GF(q)$ . The greatest common divisor of two nonzero polynomials  $f(x)$  and  $g(x)$  is defined as the monic polynomial of greatest degree that divides both  $f(x)$  and  $g(x)$ . For every pair of polynomials  $f(x)$  and  $g(x)$ , with  $g(x) \neq 0$ , there exists a unique pair of polynomials  $q(x)$  and  $r(x)$  such that

$$f(x) = q(x)g(x) + r(x) \quad (6)$$

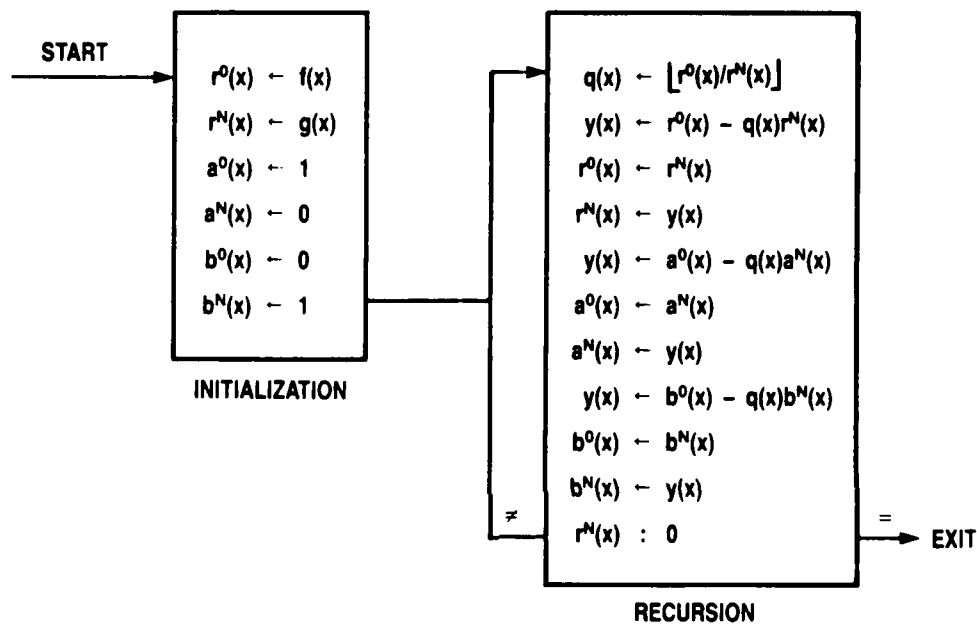
and  $\deg(r(x)) < \deg(g(x))$ . However,  $r(x)$  is not necessarily monic. At the termination of Euclid's algorithm (program 3),

$$\beta \cdot \gcd(f(x), g(x)) = r^0(x) = a^0(x)f(x) + b^0(x)g(x)$$

where  $\beta = r_0^0 \in GF(q)$  is a field element.

Program 3 is identical to program 2, except that polynomials have replaced integers in all program statements. The notation  $f(x)/g(x)$  represents the rational function or infinite power series in  $x$  obtained when the polynomial  $f(x)$  is divided by the polynomial  $g(x)$ . The notation  $[f(x)/g(x)]$  denotes the quotient polynomial  $q(x)$  of (6), i.e., the polynomial or "integer" part of the division of  $f(x)$  by  $g(x)$ . The notation  $|f(x)|_{g(x)}$  will be used to represent the remainder polynomial  $r(x)$ .





**INPUT:** POLYNOMIALS  $f(x), g(x) > 0$ ;  $\text{DEG}(f(x)) \geq \text{DEG}(g(x))$

**OUTPUT:**  $r^0(x) = \beta \cdot \text{gcd}(f(x), g(x)) = a^0(x) f(x) + b^0(x) g(x), \beta \in \text{GF}(q)$

**Program 3. EUCLID'S ALGORITHM FOR POLYNOMIALS OVER  $\text{GF}(q)$**

The recursion in program 3 is directly analogous to that of program 2. An analogy can also be drawn to equations (4), and (5) for polynomials  $a^k(x)$ ,  $b^k(x)$  and  $r^k(x)$ :

$$a^k(x)f(x) + b^k(x)g(x) = r^k(x) \quad (7)$$

$$a^k(x)b^{k-1}(x) - b^k(x)a^{k-1}(x) = (-1)^{k+1} \quad (8)$$

Equation (8) implies that  $\gcd(a^k(x), b^k(x)) = \beta$ , a field element.

Example 2:  $f(x) = x^5 + 3x^4 + 3x^2 + 5x + 10$   
 $g(x) = 2x^2 + 7x + 3$

over GF(11).

$$f(x)/g(x) = 6x^3 + 8x^2 + 7x + 9 + 10x^{-1} + 6x^{-2} + 8x^{-3} + 7x^{-4} + 2x^{-5} \\ + 10x^{-6} + 6x^{-7} + 8x^{-8} + 7x^{-9} + 2x^{-10} + \dots$$

$$\lfloor f(x)/g(x) \rfloor = 6x^3 + 8x^2 + 7x + 9 = q^1(x)$$

$$\lfloor f(x) \rfloor_{g(x)} = 9x + 5 = r^1(x)$$

Results of applying program 3 are presented in the following table:

k	$r^N(x)$	$q(x)$	$a^N(x)$	$b^N(x)$
-1	$x^5+3x^4+3x^2+5x+10$	-	1	0
0	$2x^2+7x+3$		0	1
1	$9x+5$	$6x^3+8x^2+7x+9$	1	$5x^3+3x^2+4x+2$
2	0	$10x+5$	$x+6$	$5x^4+4x+2$

$$\begin{aligned}\beta \cdot \gcd(f(x), g(x)) &= 9x + 5 \\ &= f(x) + (5x^3 + 3x^2 + 4x + 2)g(x)\end{aligned}$$

$$\beta = 9$$

$$\gcd(f(x), g(x)) = x + 3$$

Program 3 is adapted in sections 4 and 5 for solving the key equation for BCH decoding. First, a brief review of the decoding problem is presented in section 3.

## SECTION 3

### THE DECODING PROBLEM

In this section, based in part on material from Peterson [5] and Blahut [17], the decoding problem for BCH codes and the key equation are briefly reviewed. In subsequent sections various algorithms which have been proposed for solving the key equation and which are shown to be variants of Euclid's algorithm are presented. This section presupposes some familiarity with algebraic coding theory on the part of the reader. Encoding is not discussed, and knowledge of the structure and construction of generalized BCH and Reed-Solomon codes is assumed. The reader should refer to any of several excellent standard texts on algebraic coding theory, e.g., [10], [17-20], for further background material.

This section is divided into two parts. In section 3.1 the classical theory as developed by Peterson [4], Gorenstein and Zierler [6], and Forney [9] is presented. In section 3.2 a slightly different view is taken, and an alternative to Forney's formula for the error magnitudes is developed. The new formula is important for completion of the classical theory rather than for any computational advantage.

#### 3.1 THE CLASSICAL BCH DECODING THEORY

Assume a BCH code designed to correct  $t$  errors in a codeword of length  $n$ , where  $n$  is the order of the element  $\alpha$  of  $GF(q^m)$ , the finite field of  $q^m$  elements, used in defining the code,  $q$  is a power of a prime, and  $m$  is a given integer. If  $\alpha$  is a primitive

element,  $n = q^m - 1$ . Throughout this report we assume that the code is generated by a generator polynomial  $g(x)$  defined by

$$g(x) = \text{lcm}(f_1(x), f_2(x), \dots, f_{2t}(x))$$

where  $f_j(x)$  is the minimal polynomial of  $\alpha^j$  and  $\text{lcm}$  denotes the least common multiple. Let  $c(x)$  represent the transmitted codeword polynomial,  $e(x)$  be an error polynomial, and  $v(x) = c(x) + e(x)$  be the received codeword polynomial. Define  $2t$  error syndromes  $S_j$  by

$$S_j = v(\alpha^j) = c(\alpha^j) + e(\alpha^j) = e(\alpha^j) \quad (j = 1, \dots, 2t)$$

where the  $\alpha^j$  ( $j = 1, \dots, 2t$ ) are roots of the code generator polynomial  $g(x)$ .

Suppose  $v$  ( $v \leq t$ ) errors have occurred during transmission. Define  $v$  unknown error locations  $X_\ell$ , where  $X_\ell$  is the field element of  $GF(q^m)$  associated with the  $\ell^{\text{th}}$  error location in the codeword (numbered in ascending order of the error location numbers), and  $v$  unknown (unless  $q = 2$ ) error magnitudes  $Y_\ell$ , where  $Y_\ell \neq 0$  and  $Y_\ell \in GF(q)$ . For example, if  $e(x) = 6x^9 + 5x^8 + 3x^3$ , then  $v = 3$ ,  $X_1 = \alpha^3$ ,  $X_2 = \alpha^8$ ,  $X_3 = \alpha^9$ ,  $Y_1 = 3$ ,  $Y_2 = 5$ ,  $Y_3 = 6$ .

The  $2t$  syndromes are given by the  $2t$  BCH decoding equations

$$S_j = e(\alpha^j) = \sum_{i=0}^{n-1} e_i (\alpha^j)^i = \sum_{\ell=1}^v Y_\ell X_\ell^j. \quad (9)$$

The decoding problem for BCH codes is to solve this set of  $2t$  (nonlinear simultaneous) equations for the  $v$  unknown error locations  $X_\ell$  and the  $v$  unknown (if  $q > 2$ ) error magnitudes  $Y_\ell$ , given the  $2t$  syndromes  $S_j$ .

To assist in finding a solution, the error locator polynomial is defined as the monic polynomial having zeros at the inverse error locations  $X_\ell^{-1}$  for  $\ell = 1, \dots, v$ :

$$\Lambda(x) = \prod_{\ell=1}^v (1 - X_\ell x) = 1 + \sum_{i=1}^v \Lambda_i x^i \quad (10)$$

(If  $v = 0$ ,  $\Lambda(x)$  is defined as the zero-degree polynomial 1.)

Next, multiply (10) by  $Y_\ell X_\ell^{j+v}$  and set  $x = X_\ell^{-1}$ . For each  $j$  and each  $\ell$  we then have an equation

$$\begin{aligned} 0 &= Y_\ell X_\ell^{j+v} (1 + \sum_{i=1}^v \Lambda_i X_\ell^{-i}) \\ &= Y_\ell (X_\ell^{j+v} + \sum_{i=1}^v \Lambda_i X_\ell^{j+v-i}) \end{aligned}$$

Summing over  $\ell$  from 1 to  $v$ , for each  $j$

$$\begin{aligned} 0 &= \sum_{\ell=1}^v Y_\ell X_\ell^{j+v} + \sum_{i=1}^v \Lambda_i \sum_{\ell=1}^v Y_\ell X_\ell^{j+v-i} \\ &= S_{j+v} + \sum_{i=1}^v \Lambda_i S_{j+v-i} \end{aligned} \quad (11)$$

or, in matrix format,

$$\begin{bmatrix} S_1 & S_2 & S_3 & \cdots & S_{v-1} & S_v \\ S_2 & S_3 & S_4 & \cdots & S_v & S_{v+1} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ S_v & S_{v+1} & S_{v+2} & \cdots & S_{2v-2} & S_{2v-1} \end{bmatrix} \begin{bmatrix} \Lambda_v \\ \Lambda_{v-1} \\ \vdots \\ \Lambda_1 \end{bmatrix} = \begin{bmatrix} -S_{v+1} \\ -S_{v+2} \\ \vdots \\ -S_{2v} \end{bmatrix} \quad (12)$$

We can solve (12) for  $\Lambda(x)$  if the matrix of syndromes is nonsingular. The syndrome matrix in (12) can be shown to be the product of a Vandermonde matrix  $V$ , a diagonal matrix  $D$ , and the transpose of  $V$ :

$$S = \begin{bmatrix} S_1 & S_2 & \cdots & S_v \\ S_2 & S_3 & \cdots & S_{v+1} \\ \vdots & \vdots & \ddots & \vdots \\ S_v & S_{v+1} & \cdots & S_{2v-1} \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 1 & \dots & 1 \\ x_1 & x_2 & \dots & x_v \\ x_1^2 & x_2^2 & \dots & x_v^2 \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{v-1} & x_2^{v-1} & \dots & x_v^{v-1} \end{bmatrix} \begin{bmatrix} Y_1 X_1 & & & \\ & Y_2 X_2 & 0 & \\ & & \ddots & \\ 0 & & & Y_v X_v \end{bmatrix} \begin{bmatrix} 1 & x_1 & x_1^2 & \dots & x_1^{v-1} \\ 1 & x_2 & x_2^2 & \dots & x_2^{v-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_v & x_v^2 & \dots & x_v^{v-1} \end{bmatrix}$$

$$= \mathbf{V} \mathbf{D} \mathbf{V}^T.$$

The determinant of the Vandermonde matrix  $\mathbf{V}$  is given by

$$|\mathbf{V}| = \prod_{i>j} (x_i - x_j)$$

(see, e.g., Hamming [21], pp. 233-4). Since  $x_\ell \neq x_k$  for  $\ell \neq k$ ,  $|\mathbf{V}| \neq 0$ . Since  $x_\ell$  and  $Y_\ell$  are nonzero for  $\ell = 1, \dots, v$ ,  $|\mathbf{D}| \neq 0$ . Hence,  $|\mathbf{S}| \neq 0$ , and the system of equations (12) can be solved by inverting the matrix of syndromes  $\mathbf{S}$ . This forms the basis for Peterson's decoding algorithm [4,5].

There is one remaining problem. The actual number of errors  $v$  is unknown. Peterson's algorithm, therefore, initializes  $v$  as  $t$  and evaluates the determinant of the  $t \times t$  syndrome matrix  $\mathbf{S}$ . If

$|\mathbf{S}| \neq 0$ , then  $\Lambda$  is obtained as  $\mathbf{S}^{-1} \cdot (-S_{t+1}, -S_{t+2}, \dots, -S_{2t})^T$ ; if  $|\mathbf{S}| = 0$ ,  $v$  is reduced by 1 and  $\mathbf{S}$  is redefined by deletion of the last row and column as a  $v \times v$  matrix. Eventually, unless all  $S_j = 0$  (implying no transmission errors have occurred),



for some value of  $v$  we have  $|S| \neq 0$ , and  $\Lambda$  is obtained as  $(\Lambda^v, \dots, \Lambda^1)^T = S^{-1} \cdot (-S_{v+1}, \dots, -S_{2v})^T$ .

From Peterson's decoding algorithm we know that for a  $t$ -error-correcting BCH code, so long as  $v \leq t$  errors occur in the transmission of a codeword, there exists a unique error locator polynomial  $\Lambda(x)$  of degree  $v$  which is a solution to the system of equations (12). The decoding algorithms to be discussed in the succeeding sections finesse the direct inversion of the syndrome matrix, yet rely, to some extent, on Peterson's result which proves the existence and uniqueness of the solution.

To complete the decoding after finding  $\Lambda(x)$  we need to determine the error locations and, if  $q > 2$ , calculate the error magnitudes  $Y_\ell$ . Since  $\Lambda(x)$  was defined as the polynomial having zeros at the inverses of the error locations, these inverses can be obtained by evaluating  $\Lambda(x)$  for all field elements of  $GF(q^m)$ . This brute force solution was first proposed by Chien [8] and is known as a Chien search. It can be efficiently implemented as a finite field transform [17].

When  $q > 2$ , the error magnitudes can be obtained from the  $2t$  equations (9) by substituting the  $v$  determined error locations  $X_\ell$  into the first  $v$  equations and inverting the  $v \times v$  matrix

$$X = \begin{bmatrix} X_1 & X_2 & \cdots & X_v \\ X_1^2 & X_2^2 & \cdots & X_v^2 \\ \vdots & \vdots & \cdots & \vdots \\ X_1^{v-1} & X_2^{v-1} & \cdots & X_v^{v-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ X_1 & X_2 & \cdots & X_v \\ \vdots & \vdots & \cdots & \vdots \\ X_1^{v-1} & X_2^{v-1} & \cdots & X_v^{v-1} \end{bmatrix} \begin{bmatrix} X_1 & 0 & \cdots & 0 \\ 0 & X_2 & \cdots & 0 \\ \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & \cdots & X_v \end{bmatrix}$$

whose determinant is equal to  $X_1 X_2 \dots X_v \cdot |V|$ , where  $V$  is again the Vandermonde matrix, yielding  $(Y_1, Y_2, \dots, Y_v)^T = X^{-1} \cdot (S_1, S_2, \dots, S_v)^T$ . However, a more efficient algorithm has been given by Forney [9], based on the direct computation of the  $Y_\ell$  from the error locator polynomial and the error evaluator polynomial.

Let the syndrome polynomial  $S(x)$  be defined by

$$S(x) = \sum_{j=1}^{2t} S_j x^{j-1} = \sum_{j=1}^{2t} \sum_{i=1}^v Y_i X_i^j x^{j-1} \quad (13)$$

and let the error evaluator polynomial  $\Omega(x)$  be defined by

$$\Omega(x) = |S(x)\Lambda(x)|_{x^{2t}}. \quad (14)$$

Equation (14) is known as the "key equation" for BCH decoding. (It differs slightly from the form given by Berlekamp in [10], as the syndrome polynomial has been defined differently, allowing a slight simplification of Forney's formula.)

Expanding (14), we have

$$\begin{aligned}
 \Omega(x) &= \left| S(x) \Lambda(x) \right|_{x^{2t}} \\
 &= \left| \left[ \sum_{j=1}^{2t} \sum_{i=1}^v Y_i X_i^j x^{j-1} \right] \left[ \prod_{\ell=1}^v (1 - X_\ell x) \right] \right|_{x^{2t}} \\
 &= \left| \left[ \sum_{j=1}^{2t} \sum_{i=1}^v Y_i X_i X_i^{j-1} x^{j-1} \right] \left[ (1 - X_i x) \prod_{\ell \neq i} (1 - X_\ell x) \right] \right|_{x^{2t}} \\
 &= \left| \left[ \sum_{i=1}^v Y_i X_i (1 - X_i x) \sum_{j=1}^{2t} (X_i x)^{j-1} \right] \left[ \prod_{\ell \neq i} (1 - X_\ell x) \right] \right|_{x^{2t}} \\
 &= \left| \sum_{i=1}^v Y_i X_i (1 - X_i^{2t} x^{2t}) \prod_{\ell \neq i} (1 - X_\ell x) \right|_{x^{2t}} \\
 &= \sum_{i=1}^v Y_i X_i \prod_{\ell \neq i} (1 - X_\ell x). \tag{15}
 \end{aligned}$$

Note that  $\deg(\Lambda(x)) = v$ ,  $\deg(\Omega(x)) \leq v - 1$ .

Evaluating (15) at  $x = X_j^{-1}$  gives

$$\Omega(X_j^{-1}) = \sum_{i=1}^v Y_i X_i \prod_{\ell \neq i} (1 - X_\ell X_j^{-1})$$

$$= Y_j X_j \prod_{\ell \neq j} (1 - X_\ell X_j^{-1})$$

since

$$\prod_{\ell \neq j} (1 - X_\ell X_j^{-1}) = 0 \text{ for all } i \neq j.$$

But the formal derivative of the error locator polynomial is given by

$$\Lambda'(x) = - \sum_{i=1}^v X_i \prod_{\ell \neq i} (1 - X_\ell x)$$

and

$$\Lambda'(X_j^{-1}) = -X_j \prod_{\ell \neq j} (1 - X_\ell X_j^{-1}) \neq 0.$$

Therefore,

$$Y_j = \frac{\Omega(X_j^{-1})}{X_j \prod_{\ell \neq j} (1 - X_\ell X_j^{-1})} = - \frac{\Omega(X_j^{-1})}{\Lambda'(X_j^{-1})}. \quad (16)$$

Expression (16) is Forney's formula for direct calculation of the error magnitudes. If the more customary definition of  $S(x)$  as  $\sum_{j=1}^v S_j x^j$  is used, the denominator in (16) must be premultiplied by the term  $X_j$ .

### 3.2 EXTENSION OF THE CLASSICAL THEORY

In this section, an alternative to Forney's formula (16) for the error magnitudes is developed. To facilitate this discussion, the concept of reversal of a polynomial is introduced. If

$$p(x) = \sum_{j=0}^n p_j x^j$$

is a polynomial over some field, then the reversal of  $p(x)$ , denoted by  $\bar{p}(x)$ , is defined by

$$\bar{p}(x) = \sum_{j=0}^n p_{n-j} x^j = x^n p(1/x) \quad (17)$$

i.e.,  $\bar{p}(x)$  has the same coefficients as  $p(x)$  but in reverse order. Let

$$a(x) = \sum_{j=0}^{n_1} a_j x^j$$

and

$$b(x) = \sum_{j=0}^{n_2} b_j x^j.$$

By definition of polynomial multiplication, their product is given by

$$a(x)b(x) = c(x) = \sum_{j=0}^n c_j x^j$$

where

$$n = n_1 + n_2$$

and

$$c_j = \sum_{i=0}^j a_i b_{j-i}.$$

Now

$$\bar{a}(x) = \sum_{j=0}^{n_1} a_{n_1-j} x^j$$

and

$$b(x) = \sum_{j=0}^{n_2} b_{n_2-j} x^j$$

so that

$$\bar{a}(x)b(x) = d(x) = \sum_{j=0}^n d_j x^j$$

where

$$n = n_1 + n_2$$

and

$$d_j = \sum_{i=0}^j a_{n_1-i} b_{n_2-j+i}$$

i.e.,  $d_j$  is the sum of all products of the form  $a_k b_l$  whose indices  $k$  and  $l$  sum to

$$n_1 - i + n_2 - j + i = n - j.$$

But

$$\begin{aligned} \overline{a(x)b(x)} &= \bar{c}(x) \\ &= \sum_{j=0}^n c_{n-j} x^j \end{aligned}$$

where

$$c_{n-j} = \sum_{i=0}^{n-j} a_i b_{n-j-i}$$

i.e.,  $c_{n-j}$  is the sum of all products of the form  $a_k b_l$  whose indices sum to

$$i + n - j - i = n - j.$$

Hence,

$$c_{n-j} = d_j$$

and

$$\overline{a(x)b(x)} = \bar{a}(x)\bar{b}(x). \quad (18)$$

There is no relationship analogous to (18) that holds for addition. However (18) is sufficient to show that

$$\gcd(\bar{f}(x), \bar{g}(x)) = \overline{\beta \cdot \gcd(f(x), g(x))}$$

for some field element  $\beta$ . For if  $c(x) = \gcd(f(x), g(x))$  then

$$f(x) = c(x)d(x)$$

and

$$g(x) = c(x)e(x)$$

for some polynomials  $d(x)$  and  $e(x)$ , and by (18),  $\bar{c}(x)$  is a common divisor of

$$\bar{f}(x) = \bar{c}(x)\bar{d}(x)$$

and

$$\bar{g}(x) = \bar{c}(x)\bar{e}(x).$$



Similarly, if  $c'(x) = \gcd(\bar{f}(x), \bar{g}(x))$ , then  $\bar{c}'(x)$  is a common divisor of  $\bar{f}(x)$  and  $\bar{g}(x)$ . Thus

$$c'(x) = \beta \bar{c}'(x)$$

for some field element  $\beta$ .

An alternative formula to (16) can now be developed in terms of reversals. Let

$$A(x) = \lfloor (\Lambda(x)S(x))/x^{2t} \rfloor. \quad (19)$$

Then

$$\Lambda(x)S(x) = x^{2t}A(x) + Q(x). \quad (20)$$

The polynomials  $A(x)$  and  $Q(x)$  contain equivalent information in that the syndrome polynomial can be recovered either from the pair  $(\Lambda(x), A(x))$  or from the pair  $(\Lambda(x), Q(x))$ :

$$S(x) = \frac{x^{2t}A(x) + Q(x)}{\Lambda(x)} = \lfloor (x^{2t}A(x))/\Lambda(x) \rfloor \quad (21)$$

since  $\deg(\Lambda(x)) = v$  and  $\deg(Q(x)) \leq v - 1$ . Similarly,

$$\bar{S}(x) = \frac{x^{2t}\bar{Q}(x) + \bar{A}(x)}{\bar{\Lambda}(x)} = \lfloor (x^{2t}\bar{Q}(x))/\bar{\Lambda}(x) \rfloor. \quad (22)$$

The degrees of the polynomials in (21) and (22) are defined as follows. By (10),  $\Lambda(x)$  has degree  $v$ , so

$$\bar{\Lambda}(x) = x^v + \Lambda_1 x^{v-1} + \dots + \Lambda_{v-1} x + \Lambda_v.$$

The  $\deg(S(x))$  is defined as  $2t - 1$  (even when  $S_{2t} = 0$ ) and  $\deg(Q(x))$  as  $v - 1$  (even when  $Q_{v-1} = 0$ ) so that

$$\bar{S}(x) = S_1 x^{2t-1} + \dots + S_{2t}$$

and

$$\bar{Q}(x) = Q_0 x^{v-1} + Q_1 x^{v-2} + \dots + Q_{v-2} x + Q_{v-1}.$$

Finally,  $\Lambda(x)S(x)$  is defined to have degree  $v + 2t - 1$  and  $A(x)$  is defined to have degree  $v - 1$  so that

$$\bar{A}(x) = A_0 x^{v-1} + A_1 x^{v-2} + \dots + A_{v-2} x + A_{v-1}.$$

Example 3: Reed-Solomon code defined over  $GF(11)$  with  $\alpha = 2$ ,  $t = 3$ ,

$$g(x) = (x - 2)(x - 4)(x - 8)(x - 5)(x - 10)(x - 9)$$

$$= x^6 + 6x^5 + 5x^4 + 7x^3 + 2x^2 + 8x + 2.$$

Let  $c(x) = 0$ ,  $v(x) = e(x) = 6x^9 + 5x^8$ .

Then

$$S_1 = e(\alpha^1) = 5\alpha^8 + 6\alpha^9 = 5 \cdot 3 + 6 \cdot 6 = 4 + 3 = 7$$

$$S_2 = e(\alpha^2) = 5\alpha^6 + 6\alpha^8 = 5 \cdot 9 + 6 \cdot 3 = 1 + 7 = 8$$

$$S_3 = e(\alpha^3) = 5\alpha^4 + 6\alpha^7 = 5 \cdot 5 + 6 \cdot 7 = 3 + 9 = 1$$

$$S_4 = e(\alpha^4) = 5\alpha^2 + 6\alpha^6 = 5 \cdot 4 + 6 \cdot 9 = 9 + 10 = 8$$

$$S_5 = e(\alpha^5) = 5\alpha^0 + 6\alpha^5 = 5 \cdot 1 + 6 \cdot 10 = 5 + 5 = 10$$

$$S_6 = e(\alpha^6) = 5\alpha^8 + 6\alpha^4 = 5 \cdot 3 + 6 \cdot 5 = 4 + 8 = 1$$

$$S(x) = x^5 + 10x^4 + 8x^3 + x^2 + 8x + 7$$

$$\Lambda(x) = (1 - \alpha^8 x)(1 - \alpha^9 x) = (1 - 3x)(1 - 6x) = 7x^2 + 2x + 1$$

$$\begin{aligned} \Lambda(x)S(x) &= (7x^2 + 2x + 1)(x^5 + 10x^4 + 8x^3 + x^2 + 8x + 7) \\ &= 7x^7 + 6x^6 + 7 \end{aligned}$$

$$\Omega(x) = \left[ \Lambda(x)S(x) \right]_{x^2t} = 0x + 7$$

$$A(x) = \left[ (\Lambda(x)S(x))/x^2t \right] = 7x + 6$$

$$\Lambda(x) = 7x^2 + 2x + 1$$

$$\Omega(x) = 0x + 7$$

$$A(x) = 7x + 6$$

$$S(x) = \left[ ((7x + 6)x^6)/(7x^2 + 2x + 1) \right]$$

$$\bar{S}(x) = 7x^5 + 8x^4 + x^3 + 8x^2 + 10x + 1$$

$$\bar{\Lambda}(x) = x^2 + 2x + 7$$

$$\bar{\Omega}(x) = 7x + 0$$

$$\bar{A}(x) = 6x + 7$$

$$\bar{S}(x) = \left[ ((7x)x^6)/(x^2 + 2x + 7) \right]$$

Next we show that

$$\bar{\Lambda}(x) = \Lambda_{\nu} \prod_{i=1}^{\nu} (1 - X_i^{-1}x).$$

From the definition of the reversal (17) and (10)

$$\begin{aligned}
 \bar{\Lambda}(x) &= \prod_{i=1}^v (1 - X_i x) \\
 &= \prod_{i=1}^v (1 - X_i^{-1} x) && \text{by (18)} \\
 &= \prod_{i=1}^v (-X_i + x) \\
 &= \prod_{i=1}^v (-X_i)(1 - X_i^{-1} x) \\
 &= \prod_{i=1}^v (-X_i) \prod_{i=1}^v (1 - X_i^{-1} x) \\
 &= \Lambda_v \prod_{i=1}^v (1 - X_i^{-1} x). && (23)
 \end{aligned}$$

Equation (23) says that  $\bar{\Lambda}(x)$  is an unnormalized error locator polynomial for an error polynomial with errors at the inverse error locations  $X_i^{-1}$  ( $i = 1, \dots, v$ ). This will be used in deriving the new formula for error magnitudes.

It has been shown in (21) and (22) that the polynomials  $\Lambda(x)$  and  $\Omega(x)$  are equivalent in the sense that either can be used in conjunction with  $\Lambda(x)$  to obtain the syndrome polynomial  $S(x)$ . This equivalence between  $\Lambda(x)$  and  $\Omega(x)$  suggests that an alternative to Forney's formula (16) can be derived in terms of the polynomial  $\Lambda(x)$ . We now show that

$$Y_j = - \frac{\bar{A}(X_j)}{\bar{A}'(X_j)} X_j^{n-d} \quad (j = 1, \dots, v)$$

where  $d = 2t + 1$  is the designed distance of the code. Define

$$e^*(x) = \sum_{i=0}^{n-1} e_{n-i} \alpha^{(n-i)d} x^i$$

where all exponents and subscripts are taken modulo  $n$ . Let  $X^*_i$  and  $Y^*_i$  denote the error location numbers (field elements) and error magnitudes specified by  $e^*(x)$ , but, for convenience, indexed to correspond directly to  $X_i$  and  $Y_i$ . For example, if over  $GF(11)$  with  $\alpha = 2$ ,  $e(x) = 6x^9 + 5x^8 + 3x^3$ , then  $e^*(x) = 6x^7 + x^2 + 4x$  and  $X^*_1 = \alpha^7$ ,  $X^*_2 = \alpha^2$ ,  $X^*_3 = \alpha^1$ ,  $Y^*_1 = 6$ ,  $Y^*_2 = 1$ ,  $Y^*_3 = 4$ . Next, let  $S^*(x)$  denote the syndrome polynomial defined with respect to the error polynomial  $e^*(x)$ :

$$\begin{aligned} S^*_j &= e^*(\alpha^j) \\ &= \sum_{i=0}^{n-1} e_{n-i} \alpha^{(n-i)d} \alpha^{ij} \\ &= \sum_{i=0}^{n-1} e_{n-i} \alpha^{i(j-d)} \\ &= \sum_{i=0}^{n-1} e_i \alpha^{i(d-j)} \\ &= S_{d-j} \\ &= \xi_j. \end{aligned}$$

Therefore,  $S^*(x) = \bar{S}(x)$ .

Let  $\Lambda^*(x)$  denote the error locator polynomial defined w.r.t. the error polynomial  $e^*(x)$  and let  $\Omega^*(x)$  denote the corresponding error evaluator polynomial. Then

$$\Lambda^*(x) = \prod_{i=1}^v (1 - X_i^* x)$$

But

$$X_i^* = X_i^{-1}$$

so that

$$\Lambda^*(x) = \prod_{i=1}^v (1 - X_i^{-1} x) = \Lambda_v^{-1} \bar{\Lambda}(x)$$

by (23). Then

$$\begin{aligned} \Omega^*(x) &= \left[ \Lambda^*(x) S^*(x) \right]_x^{2t} \\ &= \Lambda_v^{-1} \left[ \bar{\Lambda}(x) \bar{S}(x) \right]_x^{2t} \\ &= \Lambda_v^{-1} \bar{\Lambda}(x). \end{aligned}$$

By Forney's formula

$$\begin{aligned}\gamma_j^* &= - \frac{\Omega^*(X_j^{*-1})}{\Lambda^{*'}(X_j^{*-1})} \\ &= - \frac{\bar{\Lambda}(X_j^{*-1})}{\bar{\Lambda}'(X_j^{*-1})} \\ &= - \frac{\bar{\Lambda}(X_j)}{\bar{\Lambda}'(X_j)}.\end{aligned}$$

But if  $X_j = \alpha^k$ , then

$$\begin{aligned}\gamma_j^* &= e_{n-k}^* \\ &= e_k \alpha^{kd} \\ &= \gamma_j X_j^d.\end{aligned}$$

Therefore

$$\begin{aligned}\gamma_j &= - \frac{\bar{\Lambda}(X_j)}{\bar{\Lambda}'(X_j)} (X_j^d)^{-1} \\ &= - \frac{\bar{\Lambda}(X_j)}{\bar{\Lambda}'(X_j)} X_j^{n-d}\end{aligned}\tag{24}$$

giving us an alternative to (16) for calculating the error magnitudes  $\gamma_j$ .

Example 4: Reed-Solomon 3-error-correcting code over GF(11).

Let  $\alpha = 2$ ,  $c(x) = 0$ ,  $v(x) = e(x) = 6x^9 + 5x^8 + 3x^3$ .

Then  $S(x) = 10x^5 + 7x^4 + 9x^3 + 8x^2 + 2x + 9$ .

$$\begin{array}{ll} \Lambda(x) = 10x^3 + 2x^2 + 5x + 1 & \bar{\Lambda}(x) = x^3 + 5x^2 + 2x + 10 \\ \Lambda'(x) = 8x^2 + 4x + 5 & \bar{\Lambda}'(x) = 3x^2 + 10x + 2 \\ \Omega(x) = 3x^2 + 3x + 9 & \bar{\Omega}(x) = 2x + 1 \end{array}$$

By Forney's formula (16)

$$\gamma_1 = -\frac{\Omega(\alpha^7)}{\Lambda'(\alpha^7)} = -\frac{3\alpha^4 + 3\alpha^7 + 9}{8\alpha^4 + 4\alpha^7 + 5} = -\frac{4 + 10 + 9}{7 + 6 + 5} = -\frac{1}{7} = 3$$

$$\gamma_2 = -\frac{\Omega(\alpha^2)}{\Lambda'(\alpha^2)} = -\frac{3\alpha^4 + 3\alpha^2 + 9}{8\alpha^4 + 4\alpha^2 + 5} = -\frac{4 + 1 + 9}{7 + 5 + 5} = -\frac{3}{6} = 5$$

$$\gamma_3 = -\frac{\Omega(\alpha^1)}{\Lambda'(\alpha^1)} = -\frac{3\alpha^2 + 3\alpha^1 + 9}{8\alpha^2 + 4\alpha^1 + 5} = -\frac{1 + 6 + 9}{10 + 8 + 5} = -5 = 6$$

By the new formula (24)

$$\gamma_1 = -\frac{\bar{\Lambda}(\alpha^3)}{\bar{\Lambda}'(\alpha^3)} \cdot \alpha^{3 \cdot 3} = -\frac{2\alpha^3 + 1}{3\alpha^6 + 10\alpha^3 + 2} \cdot \alpha^9 = -\frac{5 + 1}{5 + 3 + 2} \cdot 6 = -\frac{6}{10} \cdot 6 = 3$$

$$\gamma_2 = -\frac{\bar{\Lambda}(\alpha^8)}{\bar{\Lambda}'(\alpha^8)} \cdot \alpha^{8 \cdot 3} = -\frac{2\alpha^8 + 1}{3\alpha^6 + 10\alpha^8 + 2} \cdot \alpha^4 = -\frac{6 + 1}{5 + 8 + 2} \cdot 5 = -\frac{7}{4} \cdot 5 = 5$$

$$\gamma_3 = -\frac{\bar{\Lambda}(\alpha^9)}{\bar{\Lambda}'(\alpha^9)} \cdot \alpha^{9 \cdot 3} = -\frac{2\alpha^9 + 1}{3\alpha^8 + 10\alpha^9 + 2} \cdot \alpha^7 = -\frac{1 + 1}{9 + 5 + 2} \cdot 7 = -\frac{2}{5} \cdot 7 = 6$$



Example 5: BCH (15,5) 3-error-correcting code.

Let  $\alpha$  be a root of  $f(x) = x^4 + x + 1 = 0$  over  $GF(2)$ .

Let  $e(x) = x^{14} + x^{12} + x^9$ .

Then  $S(x) = \alpha^{12}x^5 + \alpha^{10}x^4 + \alpha^9x^3 + \alpha^6x^2 + \alpha^{12}x + \alpha^6$ .

$$\begin{aligned} \Lambda(x) &= \alpha^5x^3 + \alpha^{10}x^2 + \alpha^6x + 1 & \bar{\Lambda}(x) &= x^3 + \alpha^6x^2 + \alpha^{10}x + \alpha^5 \\ \Lambda^*(x) &= \alpha^5x^2 + \alpha^6 & \bar{\Lambda}^*(x) &= x^2 + \alpha^{10} + \alpha^{10} \\ \sigma(x) &= \alpha^5x^2 + \alpha^6 & \bar{\sigma}(x) &= \alpha^{10}x^2 + \alpha^9x + \alpha^2 \\ &= \Lambda^*(x) & & \end{aligned}$$

By Forney's formula (16)

$$Y_j = - \frac{\sigma(x_j^{-1})}{\Lambda^*(x_j^{-1})} = 1.$$

By the new formula (24)

$$\begin{aligned} Y_1 &= - \frac{\bar{\Lambda}(\alpha^9)}{\bar{\Lambda}^*(\alpha^9)} \cdot \alpha^{9 \cdot 8} = \frac{\alpha^{10}\alpha^{18} + \alpha^9\alpha^9 + \alpha^2}{\alpha^{18} + \alpha^{10}} \cdot \alpha^{72} \\ &= \frac{\alpha^{13} + \alpha^3 + \alpha^2}{\alpha^3 + \alpha^{10}} \cdot \alpha^{12} = \frac{\alpha^0}{\alpha^{12}} \cdot \alpha^{12} = 1 \end{aligned}$$

$$\begin{aligned} Y_2 &= - \frac{\bar{\Lambda}(\alpha^{12})}{\bar{\Lambda}^*(\alpha^{12})} \cdot \alpha^{12 \cdot 8} = \frac{\alpha^{10}\alpha^{24} + \alpha^9\alpha^{12} + \alpha^2}{\alpha^{24} + \alpha^{10}} \cdot \alpha^{96} \\ &= \frac{\alpha^4 + \alpha^6 + \alpha^2}{\alpha^9 + \alpha^{10}} \cdot \alpha^6 = \frac{\alpha^7}{\alpha^{13}} \cdot \alpha^6 = 1 \end{aligned}$$

$$\begin{aligned}
 Y_3 &= - \frac{\bar{A}(\alpha^{14})}{\bar{A}'(\alpha^{14})} \cdot \alpha^{14 \cdot 8} = \frac{\alpha^{10} \alpha^{28} + \alpha^9 \alpha^{14} + \alpha^2}{\alpha^{28} + \alpha^{10}} \cdot \alpha^{112} \\
 &= \frac{\alpha^8 + \alpha^8 + \alpha^2}{\alpha^{13} + \alpha^{10}} \cdot \alpha^7 = \frac{\alpha^2}{\alpha^9} \cdot \alpha^7 = 1
 \end{aligned}$$

The algorithms given in succeeding sections solve the key equation (14) for  $\Lambda(x)$  and, either directly or indirectly, for  $\Omega(x)$  and  $A(x)$  as well. Decoding is then completed by applying a Chien search to  $\Lambda(x)$  to determine the error locations  $X_j$  ( $j = 1, \dots, v$ ) and then employing either (16) or (24) to obtain the error magnitudes  $Y_j$  ( $j = 1, \dots, v$ ).

# SECTION 4

## THE JAPANESE DECODING ALGORITHM

In this section the decoding algorithm of Sugiyama, Kasahara, Hirasawa, and Namekawa [12], which we call the Japanese decoding algorithm, is reviewed. It is known that the key equation (14) has a solution in polynomials  $\Omega(x)$  and  $\Lambda(x)$ , with

$$\deg(\Lambda(x)) \leq t \text{ and } \deg(\Omega(x)) \leq t - 1$$

(so long as the number of errors  $v \leq t$ ). The strategy of Sugiyama, et al., is to apply Euclid's algorithm with  $g(x) = S(x)$  and  $f(x) = x^{2t}$ . Any nonzero scalar multiple of  $x^{2t}$  will do just as well, since the term  $a^k(x)f(x)$  in the Euclidean relation

$$r^k(x) = a^k(x)f(x) + b^k(x)g(x) \quad (25)$$

will disappear modulo  $x^{2t}$ . The choice  $f(x) = -x^{2t}$  will be adopted in this section. This is done simply for convenience so that at termination  $a^k(x)$  becomes the polynomial  $A(x)$  of (19). With  $f(x) = -x^{2t}$  expression (25) becomes

$$r^k(x) = -a^k(x)x^{2t} + b^k(x)S(x),$$

where  $r^k(x)$  denotes the polynomial  $r^N(x)$  defined in the  $k$ th iteration, etc. Reducing modulo  $x^{2t}$  yields

$$r^k(x) = |b^k(x)S(x)|_{x^{2t}}.$$

Thus, if Euclid's algorithm can be terminated in such a way that the degrees of  $r^k(x)$  and  $b^k(x)$  at termination satisfy the conditions for the degrees of  $\alpha(x)$  and  $\lambda(x)$ , respectively, then it provides an effective means for solving the key equation.

Recall that, in program 3,  $b^k(x)$  is the value of  $b^N(x)$  defined at the  $k^{\text{th}}$  iteration; it follows that

$$\deg(b^k(x)) = \sum_{i=1}^k \deg(q^i(x)).$$

Also,

$$q^k(x) = r^{k-2}(x)/r^{k-1}(x).$$

Thus,

$$\deg(q^k(x)) = \deg(r^{k-2}(x)) - \deg(r^{k-1}(x))$$

or

$$\begin{aligned} \deg(r^{k-1}(x)) &= \deg(r^{k-2}(x)) - \deg(q^k(x)) \\ &= \deg(r^{k-3}(x)) - (\deg(q^k(x)) + \deg(q^{k-1}(x))) \\ &\quad \vdots \\ &= \deg(r^{-1}(x)) - \sum_{i=1}^k \deg(q^i(x)) \\ &= \deg(f(x)) - \deg(b^k(x)). \end{aligned}$$

If Euclid's algorithm is terminated at the least value of  $k > 0$  for which  $\deg(r^k(x)) < t$ , then

$$\deg(r^{k-1}(x)) \geq t$$

which implies that

$$\deg(b^k(x)) = \deg(f(x)) - \deg(r^{k-1}(x)) \leq 2t - t = t.$$

Therefore,  $r^k(x)$  and  $b^k(x)$  satisfy the degree conditions for  $Q(x)$  and  $\Lambda(x)$ , respectively.

Suppose, however, that  $Q^*(x)$  and  $\Lambda^*(x)$  are another pair of solutions to (14), with

$$\deg(\Lambda^*(x)) < \deg(b^k(x)).$$

Then

$$\begin{aligned} r^k(x)\Lambda^*(x) &= -a^k(x)x^{2t}\Lambda^*(x) + b^k(x)S(x)\Lambda^*(x) \\ Q^*(x)b^k(x) &= -\Lambda^*(x)x^{2t}b^k(x) + \Lambda^*(x)S(x)b^k(x) \end{aligned} \tag{26}$$

or

$$\left| r^k(x)\Lambda^*(x) \right|_{x^{2t}} = \left| Q^*(x)b^k(x) \right|_{x^{2t}}.$$

However,

$$\deg(r^k(x)) < t, \deg(\Lambda^*(x)) \leq t \Rightarrow \deg(r^k(x)\Lambda^*(x)) < 2t$$

and

$$\deg(\Omega^*(x)) < t, \deg(b^k(x)) \leq t \Rightarrow \deg(\Omega^*(x)b^k(x)) < 2t.$$

Therefore,

$$r^k(x)\Lambda^*(x) = \Omega^*(x)b^k(x). \quad (27)$$

Combining (26) and (27),

$$a^k(x)\Lambda^*(x) = \Lambda^*(x)b^k(x) \quad (28)$$

or

$$a^k(x)/\Lambda^*(x) = b^k(x)/\Lambda^*(x) = h(x),$$

say, with

$$\deg(h(x)) > 0.$$

Then

$$a^k(x) = h(x)\Omega^*(x)$$

$$b^k(x) = h(x)\Lambda^*(x)$$

and  $h(x)$  is a common divisor of  $a^k(x)$  and  $b^k(x)$ . But by (8),  $\gcd(a^k(x), b^k(x)) = \beta$ , a scalar, leading to a contradiction. Therefore, there can be no solutions to (14) of lower degree than  $r^k(x)$  and  $b^k(x)$ . Thus,  $r^k(x)$  and  $b^k(x)$  are the desired solutions  $\Omega(x)$  and  $\Lambda(x)$ , except for possible multiplication by a scale factor  $\gamma$ .

To obtain  $\Lambda_0 = 1$ , choose  $\gamma = b_0^k$  and define

$$\Lambda(x) = \gamma^{-1}b^k(x)$$

$$\Omega(x) = \gamma^{-1}r^k(x).$$

The polynomial  $a(x)$  need not be computed in program 3 as it is not used; if it is computed, then at termination  $a^k(x)$  is a scalar multiple of the polynomial  $\Lambda(x)$  and is the part of the product  $b^k(x)S(x)$  excised by the residue taking operation:

$$a^k(x) = [(b^k(x)S(x))/x^{2t}] = \gamma\Lambda(x).$$

Program 4 can be used to solve the key equation for  $\Lambda(x)$  and  $\Omega(x)$  by the Japanese algorithm. The termination box can be omitted; it serves no useful purpose. Neither the Chien search nor the calculation of the error magnitudes by Forney's algorithm is affected by multiplication of  $\Lambda(x)$  and  $\Omega(x)$  by the same nonzero field element.

For an illustration of the execution of program 4, let us again call on the 3-error-correcting Reed-Solomon code over GF(11) with  $\alpha = 2$  that was used in examples 3 and 4.

Example 6:  $g(x) = (x - 2)(x - 4)(x - 8)(x - 5)(x - 10)(x - 9)$   
 $= x^6 + 6x^5 + 5x^4 + 7x^3 + 2x^2 + 8x + 2.$

Let  $c(x) = 0$ ,  $v(x) = e(x) = 6x^9 + 5x^8 + 3x^3.$

Then  $S(x) = 10x^5 + 7x^4 + 9x^3 + 8x^2 + 2x + 9.$

Values of  $r^N(x)$ ,  $q(x)$ , and  $b^N(x)$  are shown in the following table for successive iterations  $k$  of program 4:

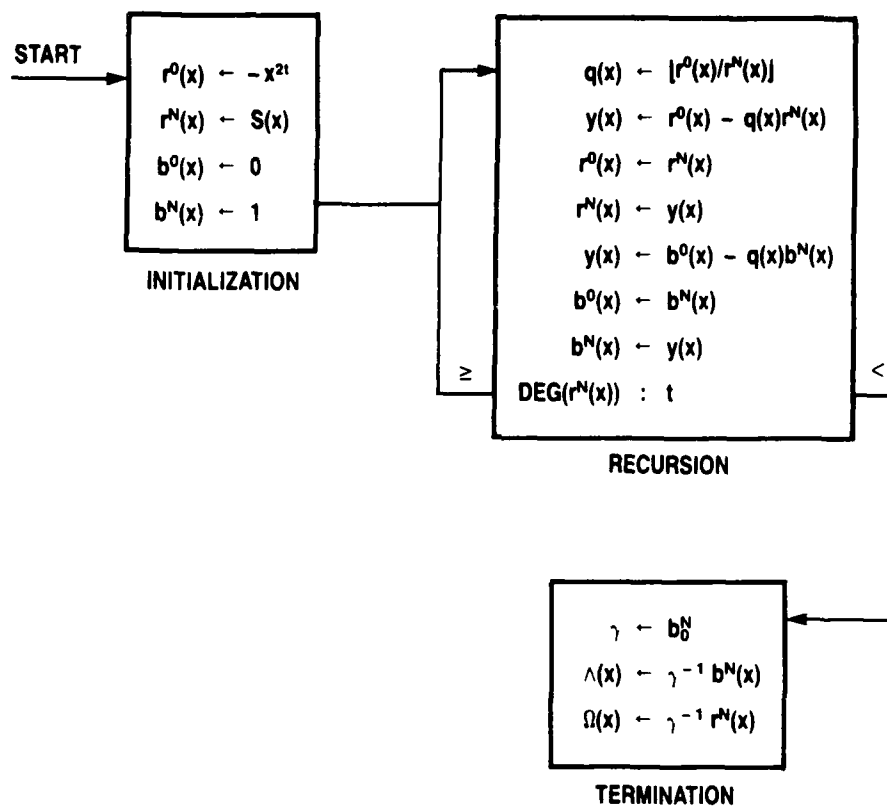
$k$	$r^N(x)$	$q(x)$	$b^N(x)$
-1	$-x^2t = -x^6$	-	0
0	$S(x) = 10x^5 + 7x^4 + 9x^3 + 8x^2 + 2x + 9$	-	1
1	$8x^4 + 6x^3 + 8x^2 + 10x + 3$	$x + 7$	$10x + 4$
2	$9x^3 + 7x^2 + 3x + 3$	$4x + 2$	$4x^2 + 8x + 4$
3	$7x^2 + 7x + 10$	$7x + 5$	$5x^3 + x^2 + 8x + 6$

$$\gamma = 6, \gamma^{-1} = 2$$

$$\Lambda(x) = 10x^3 + 2x^2 + 5x + 1 = \alpha^5 x^3 + \alpha^1 x^2 + \alpha^4 x + \alpha^0$$

$$\Omega(x) = 3x^2 + 3x + 9 = \alpha^8 x^2 + \alpha^8 x + \alpha^6.$$





INPUT: POLYNOMIALS  $x^{2t}$ ,  $S(x)$ , INTEGER  $t$

OUTPUT: POLYNOMIALS  $\wedge(x)$ ,  $\Omega(x)$   $\Omega(x) = |\wedge(x) S(x)|_{x^{2t}}$

Program 4. JAPANESE DECODING ALGORITHM

The error locations  $X_j$  ( $j = 1, \dots, 3$ ) are recovered by a Chien search on  $\Lambda(x)$ :

$$\begin{aligned}\Lambda(\alpha^0) &= \alpha^5 + \alpha^1 + \alpha^4 + \alpha^0 \\ &= 10 + 2 + 5 + 1 = 7\end{aligned}$$

$$\begin{aligned}\Lambda(\alpha^1) &= \alpha^5\alpha^3 + \alpha^1\alpha^2 + \alpha^4\alpha^1 + \alpha^0 \\ &= \alpha^8 + \alpha^3 + \alpha^5 + \alpha^0 \\ &= 3 + 8 + 10 + 1 = 0\end{aligned}$$

$$\begin{aligned}\Lambda(\alpha^2) &= \alpha^5\alpha^6 + \alpha^1\alpha^4 + \alpha^4\alpha^2 + \alpha^0 \\ &= \alpha^1 + \alpha^5 + \alpha^6 + \alpha^0 \\ &= 2 + 10 + 9 + 1 = 0\end{aligned}$$

$$\begin{aligned}\Lambda(\alpha^3) &= \alpha^5\alpha^9 + \alpha^1\alpha^6 + \alpha^4\alpha^3 + \alpha^0 \\ &= \alpha^4 + \alpha^7 + \alpha^7 + \alpha^0 \\ &= 5 + 7 + 7 + 1 = 9\end{aligned}$$

$$\begin{aligned}\Lambda(\alpha^4) &= \alpha^5\alpha^2 + \alpha^1\alpha^8 + \alpha^4\alpha^4 + \alpha^0 \\ &= \alpha^7 + \alpha^9 + \alpha^8 + \alpha^0 \\ &= 7 + 6 + 3 + 1 = 6\end{aligned}$$

$$\begin{aligned}\Lambda(\alpha^5) &= \alpha^5\alpha^5 + \alpha^1\alpha^0 + \alpha^4\alpha^5 + \alpha^0 \\ &= \alpha^0 + \alpha^1 + \alpha^9 + \alpha^0 \\ &= 1 + 2 + 6 + 1 = 10\end{aligned}$$

$$\begin{aligned}\Lambda(\alpha^6) &= \alpha^5\alpha^8 + \alpha^1\alpha^2 + \alpha^4\alpha^6 + \alpha^0 \\ &= \alpha^3 + \alpha^3 + \alpha^0 + \alpha^0 \\ &= 8 + 8 + 1 + 1 = 7\end{aligned}$$

$$\begin{aligned}\Lambda(\alpha^7) &= \alpha^5\alpha^1 + \alpha^1\alpha^4 + \alpha^4\alpha^7 + \alpha^0 \\ &= \alpha^6 + \alpha^5 + \alpha^1 + \alpha^0 \\ &= 9 + 10 + 2 + 1 = 0\end{aligned}$$

$$\begin{aligned}\Lambda(\alpha^8) &= \alpha^5\alpha^4 + \alpha^1\alpha^6 + \alpha^4\alpha^8 + \alpha^0 \\ &= \alpha^9 + \alpha^7 + \alpha^2 + \alpha^0 \\ &= 6 + 7 + 4 + 1 = 7\end{aligned}$$

$$\begin{aligned}\Lambda(\alpha^9) &= \alpha^5\alpha^7 + \alpha^1\alpha^8 + \alpha^4\alpha^9 + \alpha^0 \\ &= \alpha^2 + \alpha^9 + \alpha^3 + \alpha^0 \\ &= 4 + 6 + 8 + 1 = 8\end{aligned}$$

Therefore, the inverse error locations are given by  $\{\alpha^1, \alpha^2, \alpha^7\}$  and the error locations are given by  $\{\alpha^9, \alpha^8, \alpha^3\}$ . The error magnitudes are calculated by either Forney's formula (16) or the new formula (24) as in example 4.

To assist in comparing decoding algorithms, estimates can be made of the number of scalar multiplications required for correcting  $t$  errors. For program 4, it is assumed for this analysis that  $\deg(q(x))$  is always 1. In that case  $t$  iterations will be required. Each iteration involves two multiplications of  $r^N(x)$  by a scalar and two multiplications of  $b^N(x)$  by a scalar. The polynomial  $r^N(x)$  initially has degree  $2t - 1$  (i.e.,  $2t$  terms), and at the last iteration has degree  $t$  (i.e.,  $t + 1$  terms). The number of multiplications required to update  $r^N(x)$ , then, is

$$2 \sum_{i=t+1}^{2t} i = 2t(t + 1 + 2t)/2 = 3t^2 + t.$$

The polynomial  $b^N(x)$  initially has degree 0 (i.e., one term), and at the last iteration has degree  $t - 1$  (i.e.,  $t$  terms). The number of multiplications required to update  $b^N(x)$ , then, is

$$2 \sum_{i=1}^{2t} i = 2t(1 + t)/2 = t^2 + t.$$

(Another  $t^2 - t$  would be required if  $a(x)$  were also retained.) Program 4, then, needs on the order of  $4t^2$  multiplications for determining  $\Lambda(x)$  when  $t$  errors occur.

In addition to estimating the number of scalar multiplications it is important to provide some estimate of the number of basic time units required by the algorithm, corresponding, roughly, to the number of levels which might be required in a systolic array implementation. For this analysis it is again assumed that  $t$  errors are to be corrected and that  $\deg(q(x))$  is always 1. In this case,  $t$  iterations of the algorithm are required, each involving two multiplications of  $r^N(x)$  and  $b^N(x)$  by  $q(x) = q_1x + q_0$ . The important question to answer is whether the updates of  $r^N(x)$  and  $b^N(x)$  can proceed simultaneously. At the start of an iteration  $q_1$  is immediately available as the ratio of the leading terms of  $r^0(x)$  and  $r^N(x)$ . Thus, both the update of  $r^N(x)$  and  $b^N(x)$  can be initiated using  $q_1$ , and all terms of each polynomial can be multiplied at the same time and then subtracted from the appropriate corresponding terms of  $r^0(x)$  and  $b^0(x)$ . At completion,  $q_0$  is then available for completion of the division and polynomial updates. The algorithm thus requires  $2t$  basic time units (systolic array levels), where a basic time unit includes the time required for one finite field scalar division, one scalar multiplication, and one subtraction. The number of cells required at each level is

$$\deg(r^N(x)) + 1 + \deg(b^N(x)) + 1 = 2t + 1$$

resulting again in a total of  $4t^2 + 2t$  multiplications. This assumes that the additional cells needed for  $b^N(x)$  as it grows in length can be supplied from the cells no longer needed for  $r^N(x)$  as it shrinks.

In conclusion, then, the Japanese algorithm requires on the order of  $4t^2$  multiplications and  $2t$  basic time units for decoding  $t$  errors in a codeword of length  $n$ .

## SECTION 5

### MILLS' CONTINUED FRACTION ALGORITHM

Continued fraction approximations are closely related to Euclid's algorithm. In [13] Mills developed a continued fraction algorithm for solving linear recurrences. This algorithm can be applied to solving the key equation (14) for BCH decoding. Following the results of Welch and Scholtz [14], we show that the recursion employed in Mills' algorithm is the Euclidean recursion (1), and that, in the decoding context, the algorithm is essentially the same as the Japanese decoding algorithm with different initialization.

This section is divided into two parts. In section 5.1 the relation of continued fractions to Euclid's algorithm is explored. In section 5.2 the decoding of BCH codes by Mills' algorithm is discussed.

#### 5.1 CONTINUED FRACTIONS AND EUCLID'S ALGORITHM

Mills considers a continued fraction expansion of the form

$$\alpha = y_1 + \frac{1}{y_2 + \frac{1}{y_3 + \frac{1}{\ddots}}} \quad (29)$$

where  $y_1, y_2, y_3, \dots$ , are elements from some field. We can write

$$\begin{aligned} \alpha &= y_1 + z_1 \\ 1/z_1 &= y_2 + z_2 \\ 1/z_2 &= y_3 + z_3 \end{aligned}$$

and so forth. (In the classical case  $\alpha$  is a real number, the  $y_i$  are integers, and  $0 \leq z_i < 1$  for all  $i$ .) If  $z_m = 0$  for some  $m$ , then the continued fraction terminates with  $y_m$ . Otherwise, the continued fraction is infinite. The  $k$ th continued fraction approximation  $u_k$  to  $\alpha$  is defined by setting  $z_k = 0$ :

$$u_k = y_1 + \frac{1}{y_2 + \frac{1}{\ddots + \frac{1}{y_k}}} \quad (30)$$

Suppose  $\alpha$  is rational, i.e.,  $\alpha = s/t$  for integers  $s$  and  $t$ , with  $s > t$ . Then

$$\begin{aligned} y_1 &= \lfloor s/t \rfloor \\ z_1 &= s/t - \lfloor s/t \rfloor = (s - \lfloor s/t \rfloor t)/t \end{aligned}$$

i.e.,  $y_1$  is identical to  $q_1$  in Euclid's algorithm and  $z_1$  is identical to  $r_1/t$ . Continuing,

$$\begin{aligned} y_2 &= \lfloor 1/z_1 \rfloor = q_2 \\ z_2 &= 1/z_1 - y_2 = r_2/r_1 \end{aligned}$$

and so forth. Taking the continued fraction expansion of a rational number  $s/t$  is the same as applying Euclid's algorithm to find the  $\gcd(s,t)$ . Let us repeat example 1 from section 2 as a continued fraction expansion of  $9022/4719$ .

Example 7:  $s = 9022$ ,  $t = 4719$ .  $s/t = 1.9118457300275\dots$

$$\begin{aligned}\text{Iteration 1: } s/t &= y_1 + z_1 \\ y_1 &= \lfloor 9022/4719 \rfloor = 1 \\ z_1 &= 4303/4719 = .911845730027\dots\end{aligned}$$

The first continued fraction approximation to  $s/t$  is obtained by setting  $z_1$  equal to 0:

$$u_1 = y_1 = 1.$$

$$\begin{aligned}\text{Iteration 2: } s/t &= y_1 + 1/(y_2 + z_2) \\ y_2 &= \lfloor 1/z_1 \rfloor = \lfloor 4719/4303 \rfloor = 1 \\ z_2 &= 416/4303 = .096676737160\dots\end{aligned}$$

The second continued fraction approximation to  $s/t$  is obtained by setting  $z_2$  to 0:

$$u_2 = y_1 + 1/y_2 = 1 + 1/1 = 2.$$

$$\begin{aligned}\text{Iteration 3: } s/t &= y_1 + \frac{1}{y_2 + \frac{1}{y_3 + z_3}} \\ y_3 &= \lfloor 1/z_2 \rfloor = \lfloor 4303/416 \rfloor = 10 \\ z_3 &= 143/416 = .34375 \\ u_3 &= y_1 + \frac{1}{y_2 + \frac{1}{y_3}} = 1 + \frac{1}{1 + \frac{1}{10}} \\ &= 1 + \frac{10}{11} = \frac{21}{11} = 1.9090\dots\end{aligned}$$

$$\begin{aligned}\text{Iteration 4: } y_4 &= \lfloor 416/143 \rfloor = 2 \\ z_4 &= 130/143 = .9090\dots \\ u_4 &= 44/23 = 1.913043478260869\dots\end{aligned}$$

$$\begin{aligned}\text{Iteration 5: } y_5 &= \lfloor 143/130 \rfloor = 1 \\ z_5 &= 13/130 = .1 \\ u_5 &= 65/34 = 1.91176470588235\dots\end{aligned}$$

$$\begin{aligned}\text{Iteration 6: } y_6 &= \lfloor 130/13 \rfloor = 10 \\ z_6 &= 0 \\ u_6 &= 694/363 = s/t = 1.9118457300275\dots\end{aligned}$$

The expansion terminates at  $y_6$ . The equivalence with Euclid's algorithm shows that termination must always occur when  $\alpha$  is rational, for the  $r_k$  form a strictly decreasing sequence of nonnegative integers which must eventually, for some finite  $m$ , satisfy  $r_m = 0$ .

It may also be observed in example 7 that  $u_k = -b_k/a_k$ . That this is plausible follows from equation (4):

$$a_k s + b_k t = r_k$$

implies that

$$-b_k/a_k = s/t - r_k/(ta_k) \quad (31)$$

and

$$-a_k/b_k = t/s - r_k/(sb_k). \quad (32)$$



Thus the error, say  $e_k$ , of the approximation  $-b_k/a_k$  to  $s/t$  is given by

$$e_k = r_k / (ta_k). \quad (33)$$

When the expansion terminates with  $r_n = 0$  for some  $n$ ,  $e_n = 0$  and the final approximation  $-b_n/a_n$  is exact. Furthermore, by (5),  $\gcd(a_k, b_k) = 1$  so that  $u_n = -b_n/a_n$  is equal to  $s/t$  reduced by cancellation of all common factors, i.e.,

$$\begin{aligned} |b_n| &= s/\gcd(s, t) \\ |a_n| &= t/\gcd(s, t) \end{aligned} \quad (34)$$

More precisely, a comparison of examples 7 and 1 shows that the relationship between the approximation  $u_k$  in Mills' algorithm and the quantities  $a_k$  and  $b_k$  in the extended Euclid's algorithm can be expressed by

$$u_k = \frac{(-1)^k b_k}{(-1)^{k+1} a_k} \quad (35)$$

e.g.,

$$u_4 = \frac{44}{23} = \frac{b_4}{-a_4}.$$

Following Welch and Scholtz [14], we now show that adoption of the relation (35) leads to the Euclidean recursion (3) for  $a_k$  and  $b_k$ . Equation (29) can be written as

$$\frac{s}{t} = y_1 + \frac{1}{y_2 + \frac{1}{\ddots + \frac{1}{y_k + z_k}}} \quad (36)$$

By a process of rationalization, (36) can always be manipulated into the form

$$\frac{s}{t} = \frac{A_k(y_k + z_k) + B_k}{C_k(y_k + z_k) + D_k} \quad (37)$$

where the functions  $A_k$ ,  $B_k$ ,  $C_k$ , and  $D_k$  involve products of the  $y_j$  for  $j < k$ . Substituting  $k + 1$  for  $k$  in (37) gives

$$\frac{s}{t} = \frac{A_{k+1}(y_{k+1} + z_{k+1}) + B_{k+1}}{C_{k+1}(y_{k+1} + z_{k+1}) + D_{k+1}} \quad (38)$$

On the other hand, replacing  $z_k$  by  $(y_{k+1} + z_{k+1})^{-1}$  in (37) yields, after rationalization,

$$\frac{s}{t} = \frac{(A_k y_k + B_k)(y_{k+1} + z_{k+1}) + A_k}{(C_k y_k + D_k)(y_{k+1} + z_{k+1}) + C_k} \quad (39)$$

The  $k^{\text{th}}$  continued fraction approximation to  $s/t$  is obtained by setting  $z_k$  to 0 in (37):

$$u_k = \frac{A_k y_k + B_k}{C_k y_k + D_k} \quad (40)$$

Combining (35) and (40) we set

$$(-1)^k b_k = A_k y_k + B_k$$

$$(-1)^{k+1} a_k = C_k y_k + D_k$$

and

$$q_k = y_k$$

Then

$$\left. \begin{aligned} A_{k+1} &= A_k y_k + B_k = (-1)^k b_k \\ B_{k+1} &= A_k = (-1)^{k-1} b_{k-1} \\ y_k &= q_k \end{aligned} \right\} \Rightarrow (-1)^k b_k = (-1)^{k-1} b_{k-1} q_k + (-1)^{k-2} b_{k-2}$$

or

$$b_k = b_{k-2} - q_k b_{k-1} \quad (41)$$

$$\left. \begin{aligned} C_{k+1} &= C_k y_k + D_k = (-1)^{k+1} a_k \\ D_{k+1} &= C_k = (-1)^k a_{k-1} \\ y_k &= q_k \end{aligned} \right\} \Rightarrow (-1)^{k+1} a_k = (-1)^k a_{k-1} q_k + (-1)^{k-1} a_{k-2}$$

or

$$a_k = a_{k-2} - q_k a_{k-1} \quad (42)$$

Equations (41) and (42) are the Euclidean recursions of equation (3). Thus, taking a continued fraction expansion of a rational number  $s/t$  is identical to finding the  $\gcd(s,t)$  by Euclid's algorithm, the  $k^{\text{th}}$  continued fraction approximation being given by (35). Initial conditions are obtained as follows:

$$\begin{aligned}\frac{s}{t} &= \frac{A_1(y_1 + z_1) + B_1}{C_1(y_1 + z_1) + D_1} = y_1 + z_1 \\ \Rightarrow A_1 &= 1, B_1 = 0, C_1 = 0, D_1 = 1 \\ \Rightarrow b_1 &= -q, a_1 = 1\end{aligned}$$

$$\begin{aligned}\frac{s}{t} &= \frac{A_2(y_2 + z_2) + B_2}{C_2(y_2 + z_2) + D_2} = \frac{y_1(y_2 + z_2) + 1}{y_2 + z_2} \\ \Rightarrow A_2 &= q_1, B_2 = 1, C_2 = 1, D_2 = 0 \\ \Rightarrow b_2 &= q_1 q_2 + 1, a_2 = -q_2\end{aligned}$$

$$a_2 = a_0 - q_2 a_1 \Rightarrow a_0 = 0$$

$$a_1 = a_{-1} - q_1 a_0 \Rightarrow a_{-1} = 1$$

$$b_2 = b_0 - q_2 b_1 \Rightarrow b_0 = 1$$

$$b_1 = b_{-1} - q_1 b_0 \Rightarrow b_{-1} = 0$$

These are the same initial conditions as used for Euclid's algorithm.

Next, consider the case where  $\alpha$  of equation (29) is a ratio of two polynomials over a finite field:  $\alpha = f(x)/g(x)$ . In this case,  $\alpha$  is an infinite power series in  $x$ , the  $y_i$  are polynomials, and the  $z_i$  are infinite power series expansions in negative powers of  $x$ .

Again, as in the case where  $\alpha$  was a ratio of integers, the continued fraction must terminate for some  $m$  for which  $z_m = 0$ . We have

$$\begin{aligned} y_1 &= \lfloor f(x)/g(x) \rfloor = q^1(x) \\ z_1 &= f(x)/g(x) - \lfloor f(x)/g(x) \rfloor = r^1(x)/g(x) \\ y_2 &= \lfloor 1/z_1 \rfloor = q^2(x) \\ z_2 &= 1/z_1 - y_2 = r^2(x)/r^1(x) \end{aligned}$$

and so forth. Again, taking the continued fraction expansion of  $f(x)/g(x)$  is the same as applying Euclid's algorithm to find the  $\gcd(f(x), g(x))$ . The equivalences are exactly analogous to the integer case. Corresponding to (31) and (32) we have

$$\frac{-b^k(x)}{a^k(x)} = \frac{f(x)}{g(x)} - \frac{r^k(x)}{g(x)a^k(x)} \quad (43)$$

and

$$\frac{-a^k(x)}{b^k(x)} = \frac{g(x)}{f(x)} - \frac{r^k(x)}{f(x)b^k(x)}. \quad (44)$$

The error in approximating  $f(x)/g(x)$  by  $-b^k(x)/a^k(x)$  is given by the rational function

$$e_k = r^k(x)/(g(x)a^k(x)) \quad (45)$$

and the error in approximating  $g(x)/f(x)$  by  $-a^k(x)/b^k(x)$  by

$$e'_k = r^k(x)/(f(x)b^k(x)). \quad (46)$$

Let  $\deg(e'_k)$  denote the exponent of the first nonzero term of  $e'_k$  in (46). Then, since

$$\deg(r^k(x)) < \deg(r^{k-1}(x))$$

and

$$\deg(b^k(x)) > \deg(b^{k-1}(x))$$

it follows that

$$\deg(e'_k) \leq \deg(e'_{k-1}) - 2 \quad (47)$$

that is, the approximations (43) and (44) match at least two more terms of the series  $f(x)/g(x)$  or the series  $g(x)/f(x)$  at each iteration.

Corresponding to (35), the  $k^{\text{th}}$  continued fraction approximation to  $f(x)/g(x)$  is given by

$$u_k = \frac{(-1)^k b^k(x)}{(-1)^{k+1} a^k(x)} \quad (48)$$

and since by (8)  $\gcd(a^k(x), b^k(x)) = \gamma$  for some field element  $\gamma$ , (48) is the approximation to  $f(x)/g(x)$  with polynomials  $a^k(x)$  and  $b^k(x)$  of lowest degree possible.

Finally, by setting

$$(-1)^k b^k(x) = A_k y_k + B_k$$

$$(-1)^{k+1} a^k(x) = c_k y_k + D_k$$

and

$$q_k(x) = y_k$$

in (40), one obtains the Euclidean recursion for polynomials:

$$\begin{aligned} b^k(x) &= b^{k-2}(x) - q^k(x)b^{k-1}(x) \\ a^k(x) &= a^{k-2}(x) - q^k(x)a^{k-1}(x) \end{aligned} \tag{49}$$

For illustrative purposes, we repeat example 2 from section 2 as a continued fraction expansion of  $f(x)/g(x)$ :

$$\begin{aligned} \text{Example 8: } f(x) &= x^5 + 3x^4 + 3x^2 + 5x + 10 \\ g(x) &= 2x^2 + 7x + 3 \end{aligned}$$

over GF(11).

$$\begin{aligned} \text{Iteration 1: } f(x)/g(x) &= y_1 + z_1 \\ y_1 &= \lfloor f(x)/g(x) \rfloor = 6x^3 + 8x^2 + 7x + 9 \\ z_1 &= (9x + 5)/(2x^2 + 7x + 3) \end{aligned}$$

The first continued fraction approximation to  $f(x)/g(x)$  is obtained by setting  $z_1$  equal to 0:

$$u_1 = y_1 = 6x^3 + 8x^2 + 7x + 9 = -b^1(x)/a^1(x)$$

$$\begin{aligned} \text{Iteration 2: } f(x)/g(x) &= y_1 + 1/(y_2 + z_2) \\ y_2 &= \lfloor 1/z_1 \rfloor = 10x + 5 \\ z_2 &= 0 \\ u_2 &= y_1 + 1/y_2 = (5x^4 + 4x + 2)/(10x + 5) \\ &= b^2(x)/(-a^2(x)) \end{aligned}$$

In section 5.2 it is shown how continued fraction expansions can be used for solving the key equation.

## 5.2 DECODING BCH CODES BY MILLS' ALGORITHM

In section 5.1, the equivalence of a continued fraction expansion of a ratio of polynomials to Euclid's algorithm has been demonstrated. In this section our purpose is to show how a continued fraction expansion can be employed for solving the key equation (14). This is narrower than the question addressed by Mills in [13], which is to find a continued fraction expansion for an infinite power series defined from an arbitrary infinite sequence  $s_0, s_1, \dots$ , of elements from some field. However, the expansion we use is the same as Mills', and we shall call the resulting algorithm, which is very similar to the Japanese algorithm, Mills' decoding algorithm.

In section 3.2 it was seen that the reversal  $\bar{S}(x)$  of the syndrome polynomial is given by (22)

$$\bar{S}(x) = \frac{x^{2t}\bar{Q}(x) + \bar{A}(x)}{\bar{\Lambda}(x)}$$

so that

$$\frac{\bar{S}(x)}{x^{2t}} = \frac{\bar{Q}(x) + \bar{A}(x)/x^{2t}}{\bar{\Lambda}(x)}.$$

But, if we set  $f(x) = -x^{2t}$ ,  $g(x) = \bar{S}(x)$ , then by (44)



$$-\frac{a^k(x)}{b^k(x)} = -\frac{\tilde{S}(x)}{x^{2t}} + \frac{r^k(x)/x^{2t}}{b^k(x)}$$

or

$$\frac{\tilde{S}(x)}{x^{2t}} = \frac{a^k(x)}{b^k(x)} + \frac{r^k(x)/x^{2t}}{b^k(x)} \quad (50)$$

where the  $k^{\text{th}}$  approximation to  $\tilde{S}(x)/x^{2t}$  is given by  $a^k(x)/b^k(x)$ . For this approximation to represent a solution of the key equation, with  $\bar{Q}(x) = \gamma^{-1}a^k(x)$ ,  $\bar{A}(x) = \gamma^{-1}b^k(x)$ , and  $\bar{R}(x) = \gamma^{-1}r^k(x)$ , it is required that

$$\begin{aligned} \deg(r^k(x)) &< t \\ \deg(a^k(x)) &< t \end{aligned}$$

and

$$\deg(b^k(x)) \leq t.$$

Let  $K$  be the first index  $k$  for which  $\deg(r^k(x)) < t$ . Then, exactly as in section 4,

$$\begin{aligned} \deg(b^K(x)) &= \deg(x^{2t}) - \deg(r^{K-1}(x)) \\ &\leq 2t - t = t \end{aligned}$$

and

$$\deg(a^K(x)) < \deg(b^K(x))$$

so that  $a^K(x)$ ,  $b^K(x)$ , and  $r^K(x)$  satisfy the degree requirements. Furthermore, by (8),

$$\gcd(a^K(x), b^K(x)) = \gamma$$

for some field element  $\gamma$ , so that any other solution to (14) is some polynomial multiple of  $a^K(x)$ ,  $b^K(x)$ , and  $r^K(x)$ . Thus, the polynomials  $a^K(x)$ ,  $b^K(x)$ , and  $r^K(x)$  are the desired  $\bar{Q}(x)$ ,  $\bar{L}(x)$  and  $\bar{A}(x)$ , except for possible multiplication by a scalar.

Program 5 is a representation of a BCH decoding algorithm based upon Mills' continued fraction algorithm. The only significant differences between program 5 and program 4 are in the initialization of  $r^0(x)$  and  $r^N(x)$  and in the termination. In accordance with the definition of  $\bar{S}(x)$ , the syndrome values defining the polynomial  $r^0(x)$  are in reverse order of those defining  $r^0(x)$  in the Japanese algorithm. At termination, the program furnishes scalar multiples of the reversed polynomials  $\bar{Q}(x)$  and  $\bar{L}(x)$ .

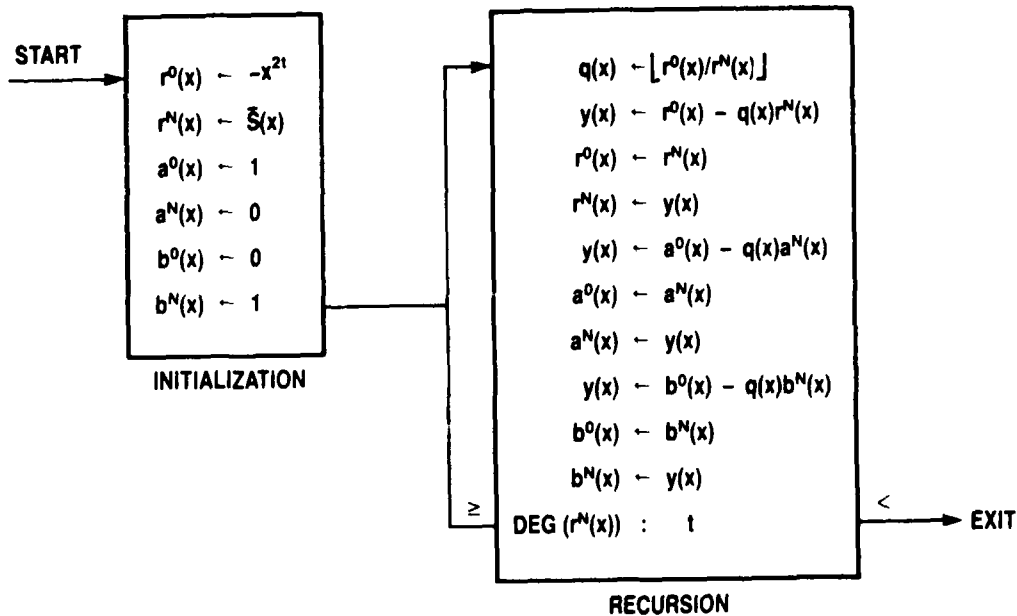
For illustration we use the same example employed in section 4, based on the 3-error-correcting Reed-Solomon code generated by

$$g(x) = x^6 + 6x^5 + 5x^4 + 7x^3 + 2x^2 + 8x + 2$$

over  $GF(11)$ .

Example 9: Let  $c(x) = 0$ ,  $v(x) = e(x) = 6x^9 + 5x^8 + 3x^3$ .

$$\text{Then } \bar{S}(x) = 9x^5 + 2x^4 + 8x^3 + 9x^2 + 7x + 10.$$



INPUT: SYNDROME POLYNOMIAL  $\tilde{S}(x)$ ; INTEGER  $t$

OUTPUT:  $\gamma_{\bar{\Lambda}}(x) = b^N(x)$ ,  $\gamma_{\bar{\Omega}}(x) = a^N(x)$

Program 5. MILLS' DECODING ALGORITHM

k	$r^N(x)$	$q^N(x)$	$a^N(x)$	$b^N(x)$
-1	$-x^6 = 10x^6$	-	1	0
0	$\tilde{S}(x) = 9x^5 + 2x^4 + 8x^3 + 9x^2 + 7x + 10$	-	0	1
1	$6x^4 + 8x^3 + 3x^2 + 8x + 6$	$6x + 6$	1	$5x + 5$
2	$4x^3 + 2x^2 + 4x + 9$	$7x + 2$	$4x + 9$	$9x^2 + 10x + 2$
3	$0x^2 + 6x + 3$	$7x + 4$	$5x^2 + 9x + 9$	$3x^3 + 4x^2 + 6x + 8$

$$\overline{a^3}(x) = 9x^2 + 9x + 5$$

$$\overline{b^3}(x) = 8x^3 + 6x^2 + 4x + 3$$

$$\gamma = 3, \gamma^{-1} = 4$$

$$\overline{r^3}(x) = 3x^2 + 6x$$

$$\Omega(x) = 3x^2 + 3x + 9$$

$$\Lambda(x) = 10x^3 + 2x^2 + 5x + 1$$

$$A(x) = x^2 + 2x$$

$$\Lambda(x)S(x) = x^8 + 2x^7 + 3x^2 + 3x + 9$$

The first approximation to  $\tilde{S}(x)/x^{2t}$  is given by  $a^1(x)/b^1(x)$ :

$$\frac{1}{5x + 5} = 9x^{-1} + 2x^{-2} + 9x^{-3} + \dots$$

The second approximation is given by  $a^2(x)/b^2(x)$ :

$$\frac{4x + 9}{9x^2 + 10x + 2} = 9x^{-1} + 2x^{-2} + 8x^{-3} + 9x^{-4} + 4x^{-5} + \dots$$

The third approximation to  $\tilde{S}(x)/x^{2t}$  is given by  $a^3(x)/b^3(x)$ :

$$\frac{5x^2 + 9x + 9}{3x^3 + 4x^2 + 6x + 8} = 9x^{-1} + 2x^{-2} + 8x^{-3} + 9x^{-4} + 7x^{-5} + 10x^{-6} + \dots$$

Note that, in accordance with (47), the approximation to  $\tilde{S}(x)/x^{2t}$  yields two additional terms at each iteration.

Program 5 is less efficient than program 4 because  $a(x)$  must now be retained. For correcting  $t$  errors, program 5 typically requires  $3t^2 + t$  multiplications for  $r(x)$ ,  $t^2 + t$  for  $b(x)$ , and  $t^2 - t$  for  $a(x)$  for a total of the order of  $5t^2$ . By dropping unneeded terms of  $r(x)$  this total can be reduced to  $4t^2$ . This is discussed further in section 7.3.

Timing for program 5 is the same as for program 4, assuming that  $a(x)$  can be updated simultaneously with  $b(x)$ . Under the assumption of  $t$  errors and  $\deg(q(x)) = 1$ , both programs require  $2t$  basic time units for their execution, where a basic time unit represents the time required for one finite field scalar division, one scalar multiplication, and one scalar subtraction.

In conclusion, Mills' algorithm and the Japanese decoding algorithm are seen to be versions of Euclid's algorithm which differ only in their initial conditions and termination rules. The Japanese algorithm gives identical results to Mills' algorithm if the order of the syndromes is reversed, and vice versa. We are really dealing with one algorithm.

## SECTION 6

### THE BERLEKAMP-MASSEY ALGORITHM

In this section the Berlekamp-Massey decoding algorithm is reviewed. The algorithm was originally formulated by Berlekamp [10] for solving the key equation (14). Massey [11] rederived the algorithm as a method for synthesizing the shortest-length linear feedback shift register which will generate a given sequence. We shall follow Massey's development.

Figure 1 is an illustration of a linear feedback shift-register (LFSR) consisting of  $L$  stages. Each input  $s_j$  to the first stage is a linear combination,

$$s_j = - \sum_{i=1}^L c_i s_{j-i}$$

specified by the feedback polynomial coefficients  $c_i$  ( $i = 1, \dots, L$ ), of the preceding  $L$  inputs  $s_{j-i}$  ( $i = 1, \dots, L$ ). For our purposes,  $c_i$  and  $s_i$  are elements of the finite field  $GF(q)$  for some  $q$ , a prime power. The output from the shift register is taken from the last ( $L^{\text{th}}$ ) stage. Thus, the first  $L$  outputs  $s_0, \dots, s_{L-1}$  are identical to the initial contents of the shift register.

An LFSR is said to generate a finite sequence  $s_0, \dots, s_{N-1}$  when this sequence coincides with the first  $N$  outputs for some initial loading of the LFSR. If  $L \geq N$ , the LFSR always generates the sequence, independent of the coefficients  $c_i$ . If  $L < N$ , the LFSR generates the sequence if and only if

$$s_j + \sum_{i=1}^L c_i s_{j-i} = 0 \quad (j = L, L+1, \dots, N-1). \quad (51)$$

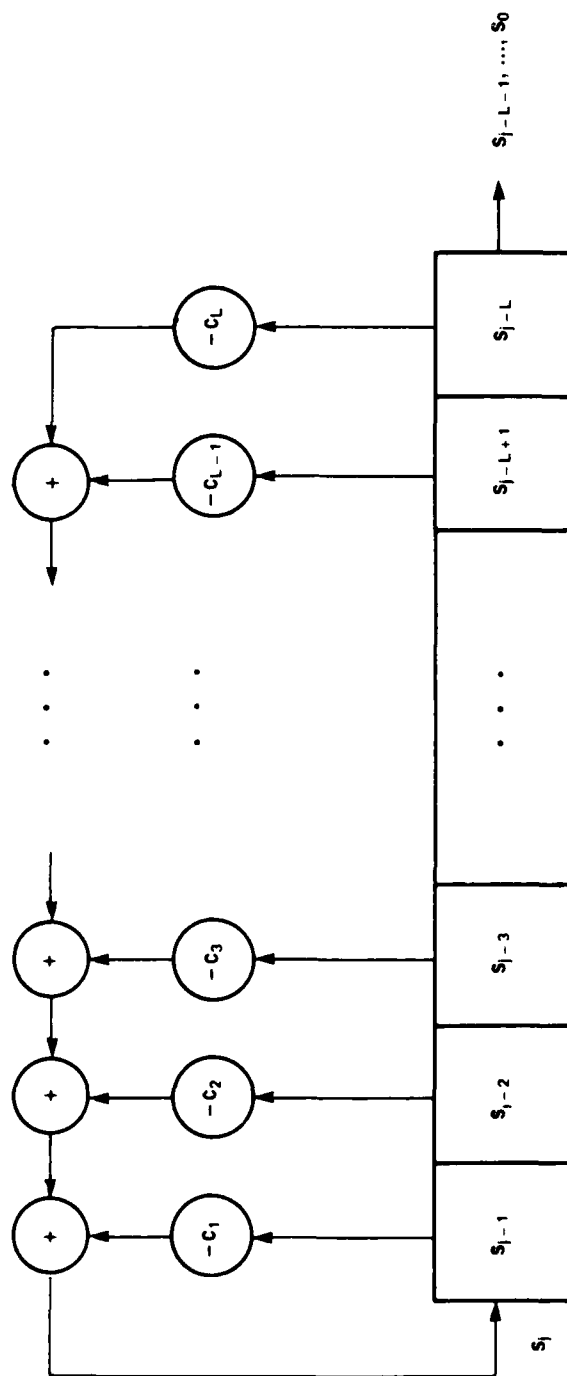


Figure 1. AN L-STAGE LINEAR FEEDBACK SHIFT REGISTER

Equation (51) is the same as equation (11) relating the syndromes  $S_j$  to the error locator polynomial coefficients  $\Lambda_i$ . Thus, for known  $\Lambda$ , equation (11) is the equation of an LFSR which generates the syndromes. We want to find a  $\Lambda(x)$  with lowest degree  $v$  (i.e., fewest errors consistent with the decoding equations). Therefore, we seek the shortest LFSR that generates the sequence of syndromes.

We begin with the statement of Massey's Theorem 1 (for a proof, see [11]).

THEOREM: If an LFSR of length  $L$  generates the sequence  $s_0, \dots, s_{N-1}$ , but not the sequence  $s_0, \dots, s_N$ , then any LFSR that generates the latter sequence must have length  $L'$  satisfying

$$L' \geq N + 1 - L.$$

Let  $s$  denote an infinite sequence  $s_0, s_1, \dots$ , and let  $L_N(s)$  denote the minimum of the lengths of all LFSR's that generate the first  $N$  symbols  $s_0, s_1, \dots, s_{N-1}$  of  $s$ . Then we have the following

COROLLARY: If some LFSR of length  $L_N(s)$  generates  $s_0, \dots, s_{N-1}$ , but not  $s_0, \dots, s_N$ , then

$$L_{N+1}(s) \geq \max(L_N(s), N + 1 - L_N(s)). \quad (52)$$

Massey's strategy is to develop an LFSR synthesis algorithm that satisfies the constraint (52) of the corollary by strict equality.



For a given sequence  $s$ , let

$$c^{(N)}(x) = 1 + \sum_{i=1}^{L_N(s)} c_i^{(N)} x^i$$

denote the connection polynomial of a minimum-length  $L_N(s)$  LFSR that generates  $s_0, \dots, s_{N-1}$ . As an inductive hypothesis, assume that  $L_N(s)$  and  $c^{(N)}(x)$  have been found for  $N = 1, 2, \dots, n$  with equality obtaining in (52) for  $N = 1, 2, \dots, n-1$ . We seek  $L_{n+1}(s)$  and  $c^{(n+1)}(x)$  with equality holding in (52) for the case  $N = n$ . From (51) we have

$$s_j + \sum_{i=1}^{L_n(s)} c_i^{(n)} s_{j-i} = \begin{cases} 0 & j = L_n(s), \dots, n-1 \\ d_n & j = n \end{cases}$$

where  $d_n$  is the discrepancy between  $s_n$  and the  $(n+1)^{\text{st}}$  symbol generated by the LFSR of length  $L_n(s)$  which generates  $s_0, \dots, s_{n-1}$ . If  $d_n = 0$ , then this LFSR also generates  $s_0, \dots, s_n$ , and  $L_{n+1}(s) = L_n(s)$  with  $c^{(n+1)}(x) = c^{(n)}(x)$ . If  $d_n \neq 0$ , a new LFSR must be found to generate  $s_0, \dots, s_n$ . We want to construct this new LFSR, with connection polynomial  $c^{(n+1)}(x)$ , to satisfy

$$L_{n+1}(s) = \max[L_n(s), n+1 - L_n(s)],$$

and

$$s_j + \sum_{i=1}^{L_{n+1}(s)} c_i^{(n+1)} s_{j-i} = 0 \quad j = L_{n+1}(s), \dots, n.$$

Massey cleverly constructs the desired LFSR by combining the latest LFSR with the LFSR which existed at the time of the last length change, using the discrepancy, say  $d_m$ , produced by that LFSR to cancel out the discrepancy  $d_n$  produced by the current LFSR.

Let  $m$  = the length of the sequence before the last LFSR length change:

$$L_m(s) < L_n(s)$$

$$s_j + \sum_{i=1}^{L_m(s)} c_i^{(m)} s_{j-i} = \begin{cases} 0 & j = L_m(s), \dots, m-1 \\ d_m \neq 0 & j = m \end{cases} \quad (53)$$

and, by hypothesis,

$$L_{m+1}(s) = L_n(s) = m + 1 - L_m(s).$$

We now rewrite (53) as

$$s_{j-n+m} + \sum_{i=1}^{L_m(s)} c_i^{(m)} s_{j-n+m-i} = \begin{cases} 0 & j = 1 - L_n(s) + n, \dots, n-1 \\ d_m \neq 0 & j = n \end{cases} \quad (54)$$

(since  $L_m(s) - m + n = 1 - L_n(s) + n$ ).

Define the new connection polynomial  $c^{(n+1)}(x)$  by

$$c^{n+1}(x) = c^{(n)}(x) - d_n d_m^{-1} x^{n-m} c^{(m)}(x). \quad (55)$$

Combining (53), (54) and (55) yields

$$\begin{aligned} s_j + \sum_{i=1}^{L_{n+1}(s)} c_i^{(n+1)} s_{j-i} \\ &= s_j + \sum_{i=1}^{L_n(s)} c_i^{(n)} s_{j-i} - d_n d_m^{-1} [s_{j-n+m} + \sum_{i=1}^{L_m(s)} c_i^{(m)} s_{j-n+m-i}] \\ &= \begin{cases} 0 & \text{for } j = L_n, L_n+1, \dots, n-1 \\ d_n & \text{for } j = n \end{cases} \quad \begin{cases} 0 & \text{for } j = 1-L_n+n, \dots, n-1 \\ d_m \neq 0 & \text{for } j = n \end{cases} \\ &= \begin{cases} 0 & \text{for } j = L_{n+1}(s), L_{n+1}(s)+1, \dots, n-1 \\ d_n - d_n d_m^{-1} d_m = 0 & \text{for } j = n \end{cases} \end{aligned} \quad (56)$$

The LFSR of length  $L_{n+1}(s) = \max(L_n(s), n+1-L_n(s))$  generates the  $n+1$  sequence symbols  $s_0, \dots, s_n$  and satisfies the constraint (52) with strict equality.

Figure 2, adapted from Blahut [17] figure 7.3, illustrates the new shift register construction.  $n - m$  new dummy stages are added to the front end of the old shift register, whose connection polynomial is  $c^{(m)}(x)$ , in order to line up its discrepancy  $d_m$  to coincide with (and cancel) the discrepancy  $d_n$  produced at  $j = n$  by the current shift register, with connection polynomial  $c^{(n)}(x)$ .

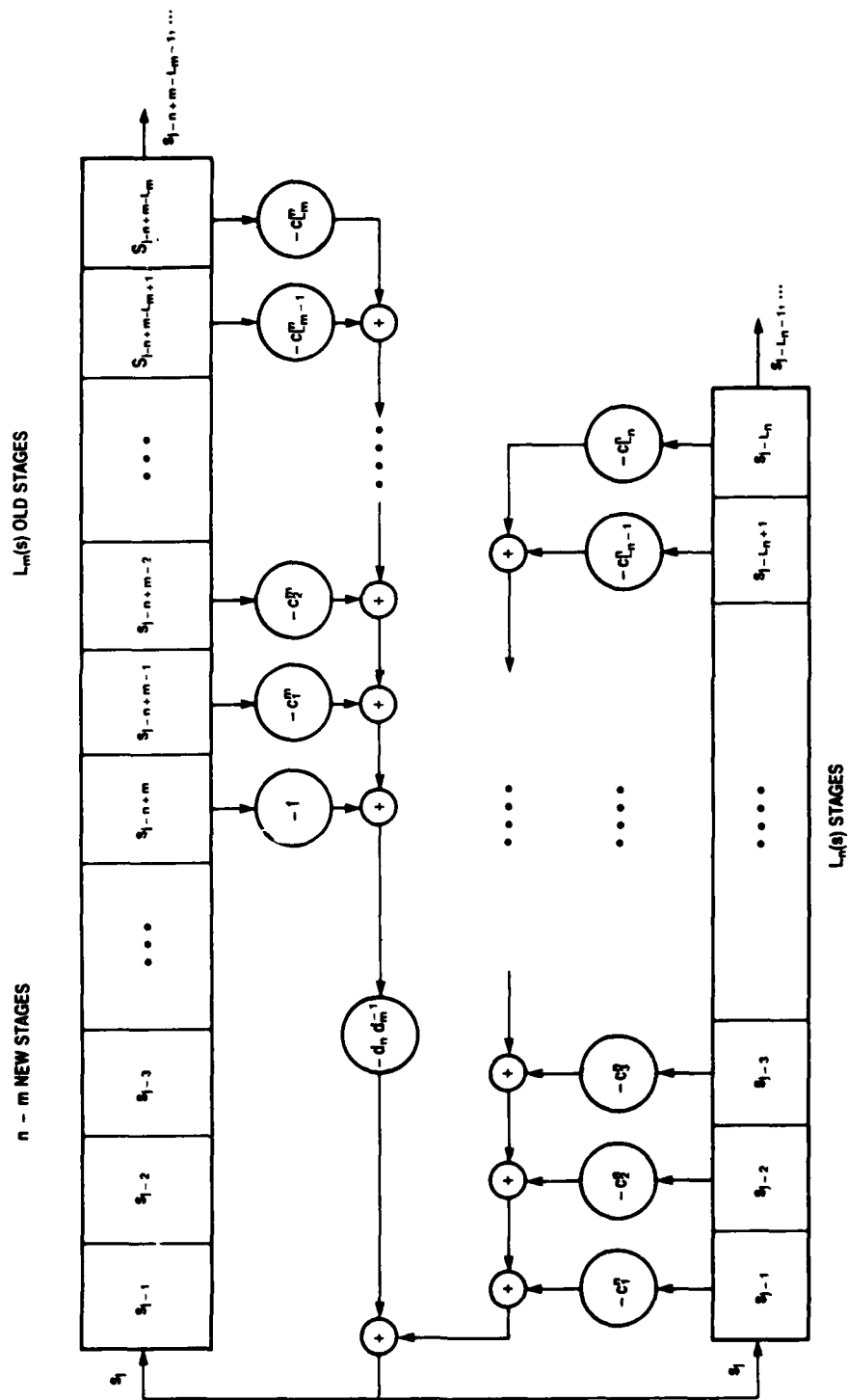


Figure 2. CONSTRUCTION OF  $c^{(n+1)}(x) = c^{(n)}(x) - d_n d_m^{-1} x^n - n c^{(n)}(x)$

When combined, the new resulting shift register will have length determined by the maximum of the length  $L_m(s)$  of the old shift register augmented by the number  $n - m$  of new stages, and the length  $L_n(s)$  of the current shift register. As can be seen directly from figure 2, the  $j^{\text{th}}$  input  $s_j$  to the combined shift register is given by

$$s_j = - \sum_{i=1}^{L_n(s)} c_i^{(n)} s_{j-i} - d_n d_m^{-1} (-s_{j-n+m} - \sum_{i=1}^{L_m(s)} c_i^{(m)} s_{j-n+m-i}).$$

Equation (56) provides a constructive proof of Massey's Theorem 2:

THEOREM: If  $L_N(s)$  denotes the length of the shortest LFSR which generates  $s_0, \dots, s_{N-1}$ , then

- (a) if some LFSR of length  $L_N(s)$  which generates  $s_0, \dots, s_{N-1}$  also generates  $s_0, \dots, s_N$ , then  $L_{N+1}(s) = L_N(s)$ ;
- (b) if some LFSR of length  $L_N(s)$  which generates  $s_0, \dots, s_{N-1}$  fails to generate  $s_0, \dots, s_N$ , then  $L_{N+1}(s) = \max(L_N(s), N + 1 - L_N(s))$ .

We observe that in case (b)

if  $L_n(s) < (n+1)/2$ , then  $L_{n+1}(s) = n + 1 - L_n(s)$ ;

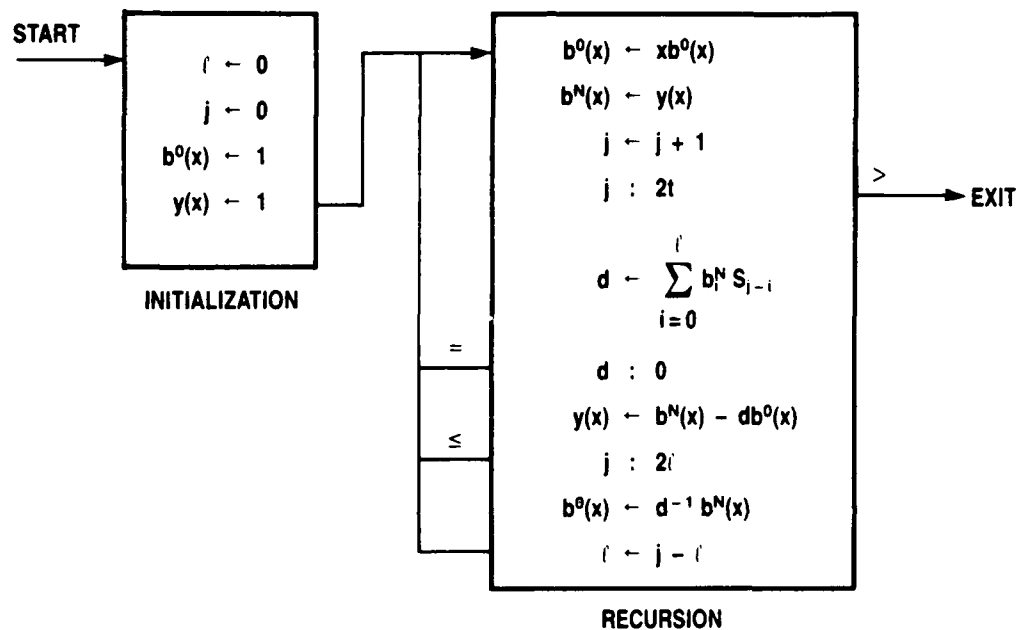
if  $L_n(s) \geq (n+1)/2$ , then  $L_{n+1}(s) = L_n(s)$ .

Program 6 is a representation of Massey's version of the Berlekamp-Massey algorithm. Inputs to the program are the syndrome polynomial  $S(x)$  and the BCH code error-correction capability  $t$ . The latter is used at line 4 of the recursion in the test for termination; the former in line 5 for calculating the discrepancy between the  $j^{\text{th}}$  syndrome and the  $j^{\text{th}}$  symbol output by the current shift register with connection polynomial  $b^N(x)$ .

The connection polynomial  $c^{(m)}(x)$  which defines the shift-register before the last length change is represented in normalized form by  $b^0(x)$ . This polynomial is defined at line 9 of the recursion by premultiplying the current  $b^N(x)$  by the inverse of the nonzero discrepancy  $d$ . (In later sections we shall consider versions of the algorithm with this normalization left out.) The polynomial  $b^0(x)$  is then updated in each iteration at line 1 of the recursion by shifting once, corresponding to the creation of a new dummy first stage, and is then ready for use in defining the new shift register connection polynomial in line 7 of the recursion.

The program path flow is slightly more complicated than for the Euclidean programs 4 and 5 as a result of the length test and branching at line 8 and the discrepancy test and branching at 6. However, both tests are, in a sense, implicit in the Euclidean programs, as will be seen in section 7.3.

At termination, the error locator polynomial  $\Lambda(x)$  is given by  $b^N(x)$ . The error evaluator polynomial  $\Omega(x)$  is not immediately available in Massey's version, and must be calculated by the key equation (14). However, as we shall see in the next section, by a slight modification of program 6 we can also obtain  $\Omega(x)$  as in the Euclidean programs.



INPUT: SYNDROME POLYNOMIAL  $S(x)$ , INTEGER  $t$

OUTPUT:  $\Lambda(x) = b^N(x)$

Program 6. BERLEKAMP-MASSEY ALGORITHM

This is an efficient algorithm in terms of the number of multiplications required. Assuming that  $\lambda$  increases by 1 at every odd numbered iteration, we require 1 multiplication to compute  $d$  at the first iteration, 2 multiplications at the second and third, 3 at the fourth and fifth, etc.,  $t - 1$  at the  $2t - 2^{\text{nd}}$  and  $2t - 1^{\text{st}}$ , and  $t$  at the  $2t^{\text{th}}$  iteration, for a total of  $t^2$  multiplications for computing the discrepancies. Assuming  $d > 0$  at all iterations, we need  $t^2 + t$  multiplications to update  $b^N(x)$  at line 7 of the recursion and  $(t^2 + t)/2$  to update  $b^0(x)$  at line 9, for a total of the order of  $2.5t^2$ . This can be reduced to  $2t^2$  by avoiding the normalization by  $d^{-1}$  in line 7, as will be discussed in section 7.2.

Program 6 has the drawback, however, that the updating of  $b(x)$  is held up until the calculation of  $d$  has been completed. This drawback is removed in the programs considered in section 7, but at the expense of requiring further multiplications. Unlike programs 4 and 5, program 6 cannot be executed in  $2t$  basic time units, and is not a candidate for implementation in a two-dimensional systolic array. At each iteration, a varying additional computation time is required to obtain the discrepancy  $d$ .

For illustration of program 6, we use the same example of a Reed-Solomon 3-error-correcting code over  $GF(11)$  as used previously for programs 4 and 5.

Example 10:  $t = 3$ ; Let  $c(x) = 0$ ,  $v(x) = e(x) = 6x^9 + 5x^8 + 3x^3$ .  
 $S(x) = 10x^5 + 7x^4 + 9x^3 + 8x^2 + 2x + 9$ .



j	l	d	$b^N(x)$	$b^0(x)$
1	0	9	1	x
2	1	9	1 + 2x	5x
3	1	10	1 + x	5x <sup>2</sup>
4	2	5	1 + x + 5x <sup>2</sup>	10x + 10x <sup>2</sup>
5	2	9	1 + 6x + 10x <sup>2</sup>	10x <sup>2</sup> + 10x <sup>3</sup>
6	3	9	1 + 6x + 8x <sup>2</sup> + 9x <sup>3</sup>	5x + 8x <sup>2</sup> + 6x <sup>3</sup>
7	3	-	1 + 5x + 2x <sup>2</sup> + 10x <sup>3</sup>	5x <sup>2</sup> + 8x <sup>3</sup> + 6x <sup>4</sup>

$$\Lambda(x) = b^N(x) = 10x^3 + 2x^2 + 5x + 1$$

$$\Omega(x) = \left| \Lambda(x)S(x) \right|_{x2t} = 3x^2 + 3x + 9$$

## SECTION 7

### HYBRIDS AND COMPARISONS

In this section, which contains the main results of this report, comparisons are made between the Berlekamp-Massey algorithm and Euclid's algorithm (in the Mills version). Hybrid programs are developed which combine features of both algorithms to advantage. The section is divided into five parts. First, the Berlekamp-Massey algorithm is expanded in a Euclidean context. Second, a new algorithm developed by Todd Citron [22] is examined and shown to belong to this same class. Third, Euclid's algorithm is modified to replace polynomial division by a sequence of partial divisions. Fourth, Mills' algorithm is modified to make it more closely resemble the Berlekamp-Massey algorithm. Fifth, comparisons are made among the resulting hybrid algorithms, which are then seen to be very similar. This similarity has been noted previously by Welch and Scholtz [14], as well as others.

#### 7.1 THE BERLEKAMP-MASSEY ALGORITHM IN EUCLIDEAN DRESS

Our objective in this section is to expand the Berlekamp-Massey algorithm in a Euclidean context. Welch and Scholtz [14] have noted a correspondence between partial results obtained for  $b^N(x)$  at certain iterations of the Berlekamp-Massey algorithm (program 6) and partial results obtained for  $b^N(x)$  at successive iterations of Mills algorithm (program 5). To explore this relationship further, we introduce polynomials  $a(x)$  and  $r(x)$  for the Berlekamp-Massey algorithm analogous to the polynomials  $a(x)$  and  $r(x)$  of Mills' algorithm.

In program 6, the next value for  $b^N(x)$  is defined at line 7 of the recursion by

$$y(x) \leftarrow b^N(x) - db^0(x). \quad (57)$$

Let us replace  $y(x)$  in (57) by a polynomial  $b^T(x)$ . In a similar fashion new values will be defined for  $a^N(x)$  and  $r^N(x)$  at each iteration by

$$\begin{aligned} a^T(x) &\leftarrow a^N(x) - da^0(x) \\ r^T(x) &\leftarrow r^N(x) - dr^0(x). \end{aligned} \quad (58)$$

At line 9 of the recursion of program 6 a new  $b^0(x)$  is defined by

$$b^0(x) \leftarrow d^{-1}b^N(x). \quad (59)$$

In a similar fashion we now define

$$\begin{aligned} a^0(x) &\leftarrow d^{-1}a^N(x) \\ r^0(x) &\leftarrow d^{-1}r^N(x). \end{aligned} \quad (60)$$

Finally, lines 1 and 2 of the recursion of program 6 will be repeated in like manner for updating  $a^0(x)$ ,  $a^N(x)$ ,  $r^0(x)$ , and  $r^N(x)$ :

$$\begin{aligned} a^0 &\leftarrow xa^0(x) \\ r^0 &\leftarrow xr^0(x) \\ a^N &\leftarrow a^T(x) \\ r^N &\leftarrow r^T(x) \end{aligned}$$

Let  $f(x)$  and  $g(x)$  be given polynomials. If initial values for  $a^0(x)$ ,  $b^0(x)$ ,  $r^0(x)$ ,  $a^N(x)$ ,  $b^N(x)$ , and  $r^N(x)$  are chosen to satisfy

$$r^0(x) = a^0(x)f(x) + b^0(x)g(x) \quad (61)$$

$$r^N(x) = a^N(x)f(x) + b^N(x)g(x) \quad (62)$$

then, by induction, (61) and (62) will continue to be satisfied at all iterations. Let  $a^k(x)$ ,  $b^k(x)$ , and  $r^k(x)$  denote the values of  $a^N(x)$ ,  $b^N(x)$ , and  $r^N(x)$  defined at the  $k^{\text{th}}$  iteration of the algorithm. Then for all  $k$

$$r^k(x) = a^k(x)f(x) + b^k(x)g(x) \quad (63)$$

for the Berlekamp-Massey algorithm. This is the same relationship that holds for Euclid's algorithm (25), even though the recursions differ. For  $f(x)$  and  $g(x)$  in (63) we shall choose

$$f(x) = -1$$

and

$$g(x) = xS(x)$$

converting (63) to

$$r^k(x) = -a^k(x) + b^k(x)xS(x), \quad k = 1, \dots, 2t. \quad (64)$$

Various initializations will work to produce the result (64).  
 To satisfy (61) we set  $r^0(x) = 1$ ,  $a^0(x) = -1$ , and  $b^0(x) = 0$ .  
 To satisfy (62) we choose  $r^N(x) = xS(x)$ ,  $a^N(x) = 0$ ,  $b^N(x) = 1$ .

In program 6,  $b^0(x)$  and  $b^N(x)$  are updated by (59) and (57). It follows that at the beginning of iteration  $j$ , with a shift register of length  $\ell$ ,

$$\ell < j \Rightarrow b_i^N = 0 \text{ for all } i > \ell. \quad (65)$$

In a similar manner, the polynomials  $a^0(x)$  and  $a^N(x)$  are updated by (58) and (60). Again, taking into consideration the initial values, we have at the beginning of iteration  $j$

$$\ell < j \Rightarrow a_i^N = 0 \text{ for all } i > \ell. \quad (66)$$

At the  $k^{\text{th}}$  iteration, let  $h^k(x) = b^k(x)xS(x)$ . Then since

$$xS(x) = \sum_{j=1}^{2t} S_j x^j = \sum_{j=0}^{2t} S_j x^j$$

if  $S_0$  is defined as 0,

$$\begin{aligned} h_k^k &= \sum_{i=0}^k b_i^k S_{k-i} \\ &= \sum_{i=0}^{\ell} b_i^k S_{k-i} \\ &= d_k \end{aligned}$$

and by (64)

$$r_k^k = -a_k^k + h_k^k = 0 + h_k^k = d_k. \quad (67)$$

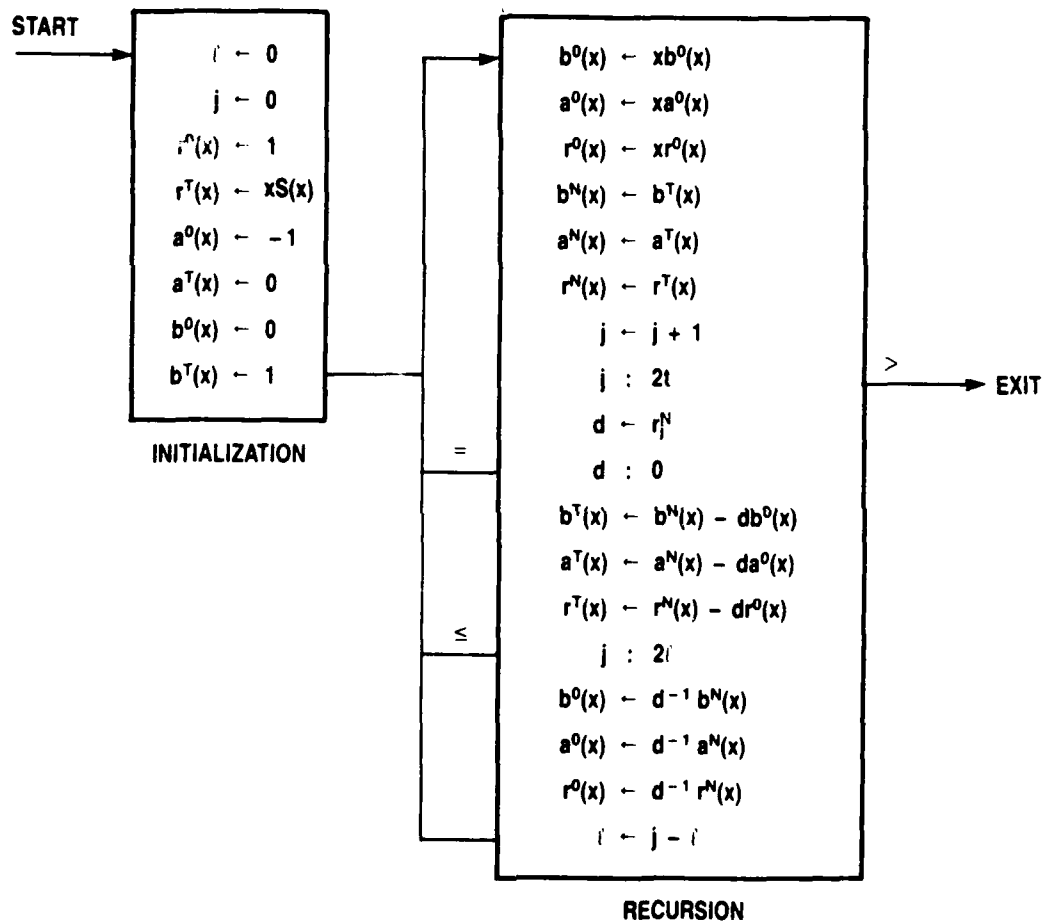
Therefore, if we keep  $r(x)$ , we eliminate the inner-product computation of the discrepancy  $d$ .

Program 7 is a representation of the Berlekamp-Massey algorithm in a Euclideanized version. We, perhaps wastefully, introduce three temporary polynomials, denoted by  $a^T(x)$ ,  $b^T(x)$ , and  $r^T(x)$ , for temporarily holding the new versions of  $a(x)$ ,  $b(x)$ , and  $r(x)$  created at lines 11-13 of the recursion. Observe that at each iteration  $j$ ,  $r_j^N = a$ . At line 17,  $r_j^0$  is set to 1; if the normalization by  $d^{-1}$  were not performed, its value would be  $d$ . After the next incrementation of  $j$ ,  $r_j^0$  is still equal to 1; if normalization were not performed at line 17 it would still have the value  $d = d_n$ . At line 13,  $r_j^T$  is set to 0. At termination,  $\Lambda(x)$  is given by  $b^N(x)$  and  $r_i^N = 0$  for  $i = 1, 2, \dots, 2t$ . Equations (14) and (64) jointly imply that

$$\Omega(x) = |(a^N(x) + r^N(x))/x|_{x^{2t}}$$

so that  $\Omega(x) = a^N(x)/x$  and  $r^N(x)/x = \Lambda(x)x^{2t}$ . Thus the expanded version of the algorithm provides both the polynomials  $\Omega(x)$  and  $\Lambda(x)$  in addition to the error locator polynomial  $\Lambda(x)$  obtained in Massey's version.

To illustrate program 7 we again use the Reed-Solomon 3-error-correcting code over  $GF(11)$  which was used to illustrate



INPUT: SYNDROME POLYNOMIAL  $S(x)$ , INTEGER  $t$

OUTPUT:  $\Lambda(x) = b^N(x)$ ,  $\Omega(x) = a^N(x)/x$

Program 7. EUCLIDEANIZED BERLEKAMP-MASSEY ALGORITHM

programs 4-6. The polynomials  $r^N(x)$ ,  $r^0(x)$ , etc., shown are those defined prior to the next incrementation of the index  $j$ , i.e. after execution of lines 1-6.

Example 11:  $t = 3$ . Let  $c(x) = 0$ ,  $v(x) = e(x) = 6x^9 + 5x^8 + 3x^3$ .  
 $S(x) = 10x^5 + 7x^4 + 9x^3 + 8x^2 + 2x + 9$ .

$j = 0, \lambda = 0$

$$\begin{aligned} r^N(x) &= xS(x) = 10x^6 + 7x^5 + 9x^4 + 8x^3 + 2x^2 + 9x \\ r^0(x) &= x \\ b^N(x) &= 1 \\ b^0(x) &= 0 \\ a^N(x) &= 0 \\ a^0(x) &= -x = 10x \end{aligned}$$

$j = 1, \lambda = 0, d = r_1^N = 9, d^{-1} = 5$

$$\begin{aligned} r^N(x) &= 10x^6 + 7x^5 + 9x^4 + 8x^3 + 2x^2 \\ r^0(x) &= 6x^7 + 2x^6 + x^5 + 7x^4 + 10x^3 + x^2 \\ b^N(x) &= 1 \\ b^0(x) &= 5x \\ a^N(x) &= 9x \\ a^0(x) &= 0 \end{aligned}$$

$j = 2, \lambda = 1, d = r_2^N = 2$

$$\begin{aligned} r^N(x) &= 10x^7 + 6x^6 + 5x^5 + 6x^4 + 10x^3 \\ r^0(x) &= 6x^8 + 2x^7 + x^6 + 7x^5 + 10x^4 + x^3 \\ b^N(x) &= x + 1 \\ b^0(x) &= 5x^2 \\ a^N(x) &= 9x \\ a^0(x) &= 0 \end{aligned}$$



$$j = 3, \ell = 1, d = r_3^N = 10, d^{-1} = 10$$

$$r^N(x) = 6x^8 + x^7 + 7x^6 + x^5 + 5x^4$$

$$r^0(x) = x^8 + 5x^7 + 6x^6 + 5x^5 + x^4$$

$$b^N(x) = 5x^2 + x + 1$$

$$b^0(x) = 10x^2 + 10x$$

$$a^N(x) = 9x$$

$$a^0(x) = 2x^2$$

$$j = 4, \ell = 2, d = r_4^N = 5$$

$$r^N(x) = x^8 + 9x^7 + 10x^6 + 9x^5$$

$$r^0(x) = x^9 + 5x^8 + 6x^7 + 5x^6 + x^5$$

$$b^N(x) = 10x^2 + 6x + 1$$

$$b^0(x) = 10x^3 + 10x^2$$

$$a^N(x) = x^2 + 9x$$

$$a^0(x) = 2x^3$$

$$j = 5, \ell = 2, d = r_5^N = 9, d^{-1} = 5$$

$$r^N(x) = 2x^9 + 0x^8 + 10x^7 + 9x^6$$

$$r^0(x) = 5x^9 + x^8 + 6x^7 + x^6$$

$$b^N(x) = 9x^3 + 8x^2 + 6x + 1$$

$$b^0(x) = 6x^3 + 8x^2 + 5x$$

$$a^N(x) = 4x^3 + x^2 + 9x$$

$$a^0(x) = 5x^3 + x^2$$

$$j = 6, \ell = 3, d = r_6^N = 9$$

$$r^N(x) = x^9 + 2x^8$$

$$r^0(x) = 5x^{10} + x^9 + 6x^8 + x^7$$

$$b^N(x) = 10x^3 + 2x^2 + 5x + 1$$

$$b^0(x) = 6x^4 + 8x^3 + 5x^2$$

$$a^N(x) = 3x^3 + 3x^2 + 9x$$

$$a^0(x) = 5x^4 + x^3$$

$j = 7$ , exit

$$\begin{aligned} \Lambda(x) &= b^N(x) = 10x^3 + 2x^2 + 5x + 1 \\ \Omega(x) &= a^N(x)/x = 3x^2 + 3x + 9 \\ x^{2t} \Lambda(x) &= r^N(x)/x = x^8 + 2x^7 \\ \Lambda(x) &= x^2 + 2x \end{aligned}$$

The price paid for retaining the polynomials  $a(x)$  and  $r(x)$  in addition to  $b(x)$  is additional multiplications. If all coefficients of  $r^N(x)$  and  $r^0(x)$  are retained, then to update  $r^T(x)$  at the first iteration takes one multiplication, and at each subsequent iteration  $j$  takes  $2t + 1 - \lfloor j/2 \rfloor$  for a total of  $3t^2$ . To update  $r^0(x)$  requires  $2t - \lfloor j/2 \rfloor$  multiplications at each odd iteration  $j$  (assuming  $\ell$  is incremented at each odd iteration) for a total of  $(3t^2 + t)/2$  multiplications. However it is not necessary to retain coefficients above those for  $x^{2t}$ . Dropping these,  $2t^2 - t + 1$  multiplications are required to update  $r^T(x)$ , and  $t^2 + t$  to update  $r^0(x)$ . In addition,  $t^2 + t$  multiplications are required to update  $b^T(x)$ ,  $t^2 - t$  for  $a^T(x)$ ,  $(t^2 + t)/2$  for  $b^0(x)$ , and  $(t^2 - t)/2$  for  $a^0(x)$ , giving a total on the order of  $6t^2$ .

The updates of all polynomials can now be performed in parallel, however, so that processing time is now  $2t$  basic time units, as for programs 4 and 5, where a basic time unit represents the time required for one finite field multiplication and one subtraction plus (half the time) one finite field inversion.

## 7.2 CITRON'S ALGORITHM

Todd Citron has presented a new algorithm [22], based on Kung's generalized Lanczos recursion [23,24] and related to Schur's algorithm [25,26], for solving the key equation (14). In this section we shall not attempt to reproduce Citron's derivation, but shall simply state Citron's algorithm using his notation and show that it belongs to the class of Euclideanized Berlekamp-Massey algorithms developed in section 7.1.

Citron describes his algorithm using matrix notation instead of employing polynomials over finite fields. The elements of his matrices correspond to the coefficients of our polynomials, as will be shown. Citron retains three two-columned arrays:  $R(i)$ , corresponding to  $r(x)$ ,  $P(i)$ , corresponding to  $b(x)$ , and  $T(i)$ , corresponding to  $a(x)$ . Citron's algorithm is stated as follows:

Recursion: At  $i^{\text{th}}$  iteration

$$\begin{bmatrix} z^{-1}R_2(i) \\ R_1(i) \end{bmatrix} = \begin{bmatrix} z^{-1} & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & -\rho_i \\ \gamma_i & (1-\gamma_i) \end{bmatrix} \begin{bmatrix} z^{-1}R_2(i-1) \\ R_1(i-1) \end{bmatrix} \quad (68)$$

$$\begin{bmatrix} z^{-1}P_2(i) \\ P_1(i) \end{bmatrix} = \begin{bmatrix} z^{-1} & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & -\rho_i \\ \gamma_i & (1-\gamma_i) \end{bmatrix} \begin{bmatrix} z^{-1}P_2(i-1) \\ P_1(i-1) \end{bmatrix} \quad (69)$$

$$\begin{bmatrix} z^{-1}T_2(i) \\ T_1(i) \end{bmatrix} = \begin{bmatrix} z^{-1} & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & -\rho_i \\ \gamma_i & (1-\gamma_i) \end{bmatrix} \begin{bmatrix} z^{-1}T_2(i-1) \\ T_1(i-1) \end{bmatrix} \quad (70)$$

where  $z^{-1}$  denotes a left-shift operation (one place),

$$\rho_i = \text{"top element" of } z^{-1}R_2(i-1) / \text{"top element" of } R_1(i-1), \quad (71)$$

$$N_i = \bar{N}_{i-1} + 1$$

$$\bar{N}_i = \begin{cases} -N_i & \text{if } N_i > 0 \text{ and } \rho_i \neq 0 \\ N_i & \text{otherwise} \end{cases}$$

$$\text{and } \gamma_i = \begin{cases} 1 & \text{if } N_i > 0 \text{ and } \rho_i \neq 0 \\ 0 & \text{otherwise} \end{cases} \quad (72)$$

Initialization:

$$\bar{N}_0 = 0$$

$$z^{-1}R_2(0) = [S_1 \ S_2 \ \dots \ S_{2t}]^T$$

$$R_1(0) = [1 \ 0 \ \dots \ 0]^T \quad (73)$$

$$z^{-1}P_2(0) = [0 \ 0 \ \dots \ 0 \ 1]^T$$

$$P_1(0) = [0 \ \dots \ 0]^T \quad (74)$$

$$z^{-1}T_2(0) = [0 \ \dots \ 0]^T$$

$$T_1(0) = [0 \ \dots \ 0 \ -1]^T \quad (75)$$

At termination:

$$z^{-1}P_2(2t) = [\Lambda_0 \ \Lambda_1 \ \dots \ \Lambda_{2t}]^T$$

$$z^{-1}T_2(2t) = [\Omega_0 \ \Omega_1 \ \dots \ \Omega_{2t}]^T \quad (76)$$

Let us first apply Citron's algorithm to our Reed-Solomon 3-error-correcting code (over GF(11)) example. Then we shall show that Citron's algorithm is a slight modification of the Euclideanized Berlekamp-Massey algorithm of program 7. For the example, we display Citron's columns as rows. By (73) we need  $2t$  places for  $R(i)$ , and starting from (74) and (75) we need  $2t + 1$  places for  $P(i)$  and  $T(i)$ , since  $2t$  left shifts will be made by the algorithm employing (69) and (70).

Example 12:  $t = 3$ ;  $c(x) = 0$ ,  $v(x) = e(x) = 6x^9 + 5x^8 + 3x^3$ .  
 $S(x) = 10x^5 + 7x^4 + 9x^3 + 8x^2 + 2x + 9$ .

$$z^{-1}R_2(0) = (9, 2, 8, 9, 7, 10) \\ R_1(0) = (1, 0, 0, 0, 0, 0)$$

$$i = 1: \quad \rho_1 = 9/1 = 9 \\ N_1 = 1 \\ \bar{N}_1 = -1 \\ \gamma_1 = 1$$

$$z^{-1}R_2(1) = (2, 8, 9, 7, 10, 0) \\ R_1(1) = (9, 2, 8, 9, 7, 10) \\ z^{-1}P_2(1) = (0, 0, 0, 0, 0, 1, 0) \\ P_1(1) = (0, 0, 0, 0, 0, 0, 1)$$

$$z^{-1}T_2(1) = (0, 0, 0, 0, 0, 9, 0) \\ T_1(1) = (0, 0, 0, 0, 0, 0, 0)$$

$$i = 2: \quad \rho_2 = 2/9 = 10 \\ N_2 = 0 \\ \bar{N}_2 = 0 \\ \gamma_2 = 0$$

$$z^{-1}R_2(2) = (10, 6, 5, 6, 10, 0)$$

$$R_1(2) = (9, 2, 8, 9, 7, 10)$$

$$z^{-1}P_2(2) = (0, 0, 0, 0, 1, 1, 0)$$

$$P_1(2) = (0, 0, 0, 0, 0, 0, 1)$$

$$z^{-1}T_2(2) = (0, 0, 0, 0, 9, 0, 0)$$

$$T_1(2) = (0, 0, 0, 0, 0, 0, 0)$$

$$i = 3: \quad \rho_3 = 10/9 = 6$$

$$N_3 = 1$$

$$\bar{N}_3 = -1$$

$$\gamma_3 = 1$$

$$z^{-1}R_2(3) = (5, 1, 7, 1, 6, 0)$$

$$R_1(3) = (10, 6, 5, 6, 10, 0)$$

$$z^{-1}P_2(3) = (0, 0, 0, 1, 1, 5, 0)$$

$$P_1(3) = (0, 0, 0, 0, 1, 1, 0)$$

$$z^{-1}T_2(3) = (0, 0, 0, 9, 0, 0, 0)$$

$$T_1(3) = (0, 0, 0, 0, 9, 0, 0)$$

$$i = 4: \quad \rho_4 = 5/10 = 6$$

$$N_4 = 0$$

$$\bar{N}_4 = 0$$

$$\gamma_4 = 0$$

$$z^{-1}R_2(4) = (9, 10, 9, 1, 0, 0)$$

$$R_1(4) = (10, 6, 5, 6, 10, 0)$$

$$z^{-1}P_2(4) = (0, 0, 1, 6, 10, 0, 0)$$

$$P_1(4) = (0, 0, 0, 0, 1, 1, 0)$$

$$z^{-1}T_2(4) = (0, 0, 9, 1, 0, 0, 0)$$

$$T_1(4) = (0, 0, 0, 0, 9, 0, 0)$$

$$\begin{aligned}
 i = 5: \quad \rho_5 &= 9/10 = 2 \\
 N_5 &= 1 \\
 \bar{N}_5 &= -1 \\
 \gamma_5 &= 1
 \end{aligned}$$

$$\begin{aligned}
 z^{-1}R_2(5) &= (9, 10, 0, 2, 0, 0) \\
 R_1(5) &= (9, 10, 9, 1, 0, 0)
 \end{aligned}$$

$$\begin{aligned}
 z^{-1}P_2(5) &= (0, 1, 6, 8, 9, 0, 0) \\
 P_1(5) &= (0, 0, 1, 6, 10, 0, 0)
 \end{aligned}$$

$$\begin{aligned}
 z^{-1}T_2(5) &= (0, 9, 1, 4, 0, 0, 0) \\
 T_1(5) &= (0, 0, 9, 1, 0, 0, 0)
 \end{aligned}$$

$$\begin{aligned}
 i = 6: \quad \rho_6 &= 9/9 = 1 \\
 N_6 &= 0 \\
 \bar{N}_6 &= 0 \\
 \gamma_6 &= 0
 \end{aligned}$$

$$\begin{aligned}
 z^{-1}R_2(6) &= (0, 2, 1, 0, 0, 0) \\
 R_1(6) &= (9, 10, 9, 1, 0, 0)
 \end{aligned}$$

$$\begin{aligned}
 z^{-1}P_2(6) &= (1, 5, 2, 10, 0, 0, 0) = (\Lambda_0, \dots, \Lambda_{2t}) \\
 P_1(6) &= (0, 0, 1, 6, 10, 0, 0)
 \end{aligned}$$

$$\begin{aligned}
 z^{-1}T_2(6) &= (9, 3, 3, 0, 0, 0, 0) = (\Omega_0, \dots, \Omega_{2t}) \\
 T_1(6) &= (0, 0, 9, 1, 0, 0, 0)
 \end{aligned}$$

A comparison with example 11 shows that the components of  $z^{-1}R_2(i)$ ,  $z^{-1}P_2(i)$ , and  $z^{-1}T_2(i)$ , are the same as the coefficients of  $r^N(x)$ ,  $b^N(x)$ , and  $a^N(x)$  obtained at the  $i^{\text{th}}$  iteration of program 7.

We now assert, and later shall show, that at each iteration  $i$ , Citron's  $\alpha_i \neq 0$  if and only if Massey's  $i^{\text{th}}$  discrepancy  $d_i \neq 0$ . With this assertion we show that Citron's variable  $N_i = i - 2^{\ell_{i-1}}$ , where  $\ell_{i-1}$  is the length of the shift-register at the beginning of the  $i^{\text{th}}$  iteration of the Berlekamp-Massey algorithm.

Lemma:  $N_i(\text{Citron}) = i - 2^{\ell_{i-1}} (\text{Massey})$

Proof: by induction on  $i$

A) Let  $i = 1$

Citron:

$$N_0 = 0$$

$$N_1 = N_0 + 1 = 1$$

Massey:

$$\ell_0 = 0$$

$$1 - 2^{\ell_0} = 1$$

Therefore, for  $i = 1$ ,  $N_i = i - 2^{\ell_{i-1}}$

b) Assume lemma is true for  $i = n - 1$ :

$$N_{n-1} = n - 1 - 2^{\ell_{n-2}}$$

Case 1:  $N_{n-1} > 0$  and  $\alpha_{n-1} \neq 0 \Rightarrow 2^{\ell_{n-2}} < n-1$  and  $d_{n-1} \neq 0$



<p>Citron:</p> $\bar{N}_{n-1} = -N_{n-1} = -n + 1 + 2\ell_{n-2}$ $N_n = \bar{N}_{n-1} + 1 = -n + 2 + 2\ell_{n-2}$	<p>Massey: length change required</p> $\ell_{n-1} = n - 1 - \ell_{n-2}$ $n - 2\ell_{n-1} = n - 2(n-1-\ell_{n-2})$ $= -n + 2 + 2\ell_{n-2}$
---	--

Case 2:  $N_{n-1} \leq 0$  or  $\rho_{n-1} = 0 \Rightarrow 2\ell_{n-2} \geq n-1$  or  $d_{n-1} = 0$

<p>Citron</p> $\bar{N}_{n-1} = N_{n-1}$ $N_n = \bar{N}_{n-1} + 1 = N_{n-1} + 1$	<p>Massey: no length change</p> $\ell_{n-1} = \ell_{n-2}$ $N - 2\ell_{n-1} = n - 2\ell_{n-2} = N_{n-1} + 1$
---	---

Therefore, truth of the lemma for  $i = n - 1$  implies truth for  $i = n$ .

Therefore, by induction on  $i$ , the lemma must be true for all  $i$ .

This result implies that for all iterations  $i$  Citron's definition of  $\gamma_i$  is equivalent to

$$\gamma_i = \begin{cases} 1 & \text{if } o_i \neq 0 \text{ and } 2\ell < i \\ 0 & \text{otherwise} \end{cases} \quad (77)$$

where  $\ell$  is Massey's shift-register length (not explicitly computed in Citron's algorithm).  $\gamma_i$  is a logical variable employed by Citron to eliminate branching. This may be important for VLSI implementation of the algorithm, but to some extent hides what is going on. For purposes of comparing algorithms we leave the branching explicit in our programs.

The first matrix on the rhs of (68) produces  $[z^{-1}R_2(i), R_1(i)]^T$  on the lhs instead of  $[R_2(i), R_1(i)]^T$ . Removing this matrix multiplication to get at the recursion proper, we have

$$R_2(i) = z^{-1}R_2(i-1) - \rho_i R_1(i-1) \quad (78)$$

and

$$R_1(i) = \begin{cases} z^{-1}R_2(i-1) & \text{if } \rho_i \neq 0 \text{ and } 2^p < i \\ R_1(i-1) & \text{otherwise} \end{cases} \quad (79)$$

Citron shifts the arrays  $R_2(i)$ ,  $P_2(i)$ , and  $T_2(i)$  at each iteration one place to the left, but does not shift  $R_1(i)$ ,  $P_1(i)$ , and  $T_1(i)$ . On the other hand, in program 7, the polynomial coefficients for  $r^0(x)$ ,  $b^0(x)$ , and  $a^0(x)$  are shifted once to the right at each iteration (lines 1-3 of the recursion) while  $r^N(x)$ ,  $b^N(x)$ , and  $a^N(x)$  are not shifted. Thus, the components of  $R_2(i)$ , etc., bear the same relation to the components of  $R_1(i)$ , etc., as the coefficients of  $r^N(x)$ , etc., to the coefficients of  $r^0(x)$ , etc. (Note, however, that at the  $i$ th iteration, the component of  $R_2(i)$  which corresponds to  $r_i^N$  is now at the "top" of the array.)

In order to compare Citron's algorithm with program 7 we should like to remove his left shift of  $R_2(i)$  and insert in its stead a right shift of  $R_1(i)$ . First, let us multiply relations (78) and (79) by the right shift operator  $z$ , yielding

$$zR_2(i) = R_2(i-1) - \rho_i zR_1(i-1) \quad (80)$$

and

$$zR_1(i) = \begin{cases} R_2(i-1) & \text{if } \rho_i \neq 0 \text{ and } 2\lambda < i \\ zR_1(i-1) & \text{otherwise} \end{cases} \quad (81)$$

Relations (80) and (81) are still the fundamental equations defining  $R(i)$  in Citron's algorithm. The  $z$  preceding  $R_2(i)$  on the lhs of equation (80) effects the left shift of the vector. To remove the left shift, we simply remove this  $z$ , producing the relation

$$R_2(i) = R_2(i-1) - \rho_i z R_1(i-1) \quad (82)$$

To induce a right shift of the vector  $R_1(i)$  we remove the  $z$  preceding  $R_1(i)$  on the lhs of equation (81), yielding

$$R_1(i) = \begin{cases} R_2(i-1) & \text{if } \rho_i \neq 0 \text{ and } 2\lambda < i \\ zR_1(i-1) & \text{otherwise} \end{cases} \quad (83)$$

Equations (82) and (83), together with a corresponding set of modifications to (69) and (70), define Citron's algorithm with the shifts changed to correspond to the shift in the Berlekamp-Massey algorithm.

But equations (82) and (83) are essentially the same equations as those found in the Berlekamp-Massey algorithm of program 7, with  $R_2(i)$  corresponding to (and identical to)  $r^N(x)$  and  $R_1(i)$  corresponding to (but not identical to)  $r^0(x)$ . When a length change is required, Citron does not define  $R_1(i)$  as  $d_m^{-1}R_2(i-1)$ ,

as on line 17 of program 7, but simply as  $R_2(i-1)$ . Therefore, when  $R_2$  is updated, it must be updated not as  $R_2(i-1) - d_n z R_1(i-1)$ , but as  $R_2(i-1) - d_n d_m^{-1} z R_1(i-1)$ . Thus,  $\rho_i$  in Citron's algorithm is the ratio of the current  $d_n$ , given by the top element of  $R_2(i-1)$ , divided by the old  $d_m$ , given by the top element of  $R_1(i-1)$ , and  $\rho_i \neq 0$  if and only if  $d_n \neq 0$ , as asserted.

Program 8 is a representation of the shifted version of Citron's algorithm employing (82) and (83) with branching (at lines 10 and 14 of the recursion) shown explicitly. We now repeat our Reed-Solomon GF(11) example for program 8. A comparison of the polynomials  $r^N(x)$ ,  $r^0(x)$ ,  $b^N(x)$ ,  $b^0(x)$ ,  $a^N(x)$ , and  $a^0(x)$  with Citron's arrays  $z^{-1}R_2(i)$ ,  $R_1(i)$ ,  $z^{-1}P_2(i)$ ,  $P_1(i)$ ,  $z^{-1}T_2(i)$ , and  $T_1(i)$  shows them to be identical (except for shifts and directions).

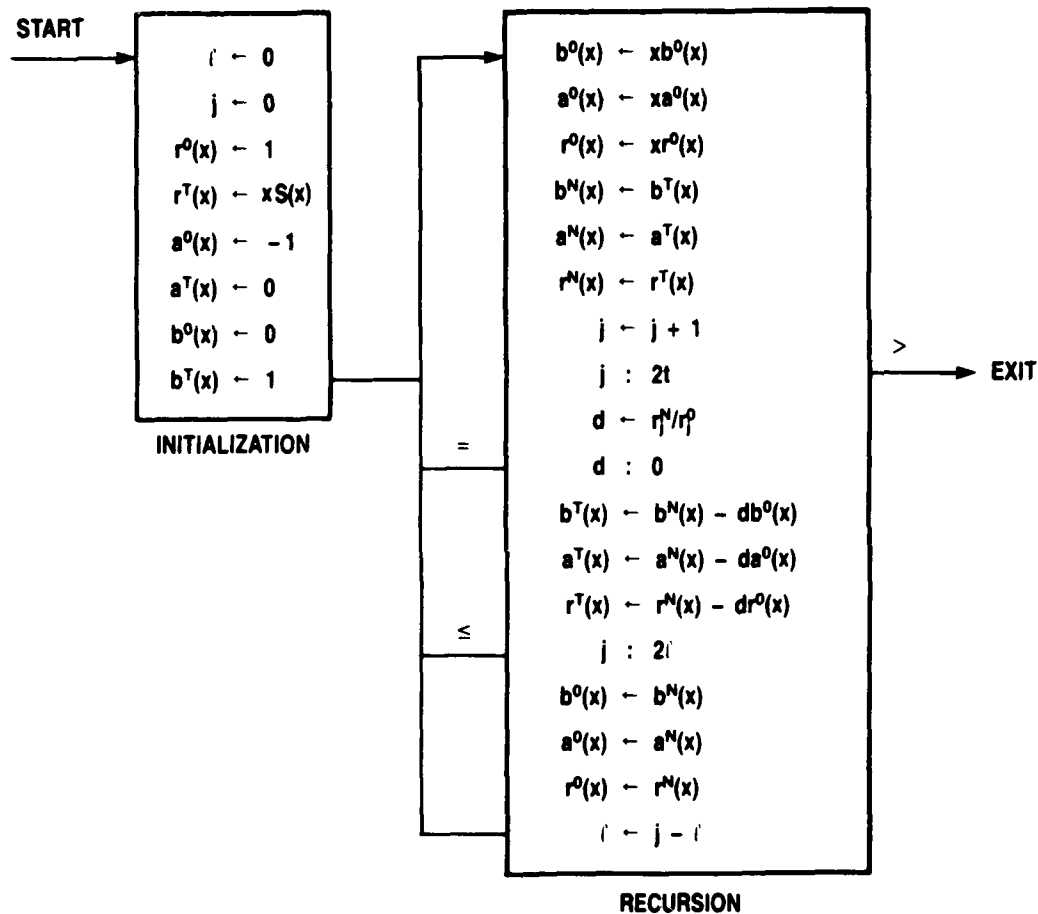
Example 13:  $t = 3$ ;  $c(x) = 0$ ,  $v(x) = e(x) = 6x^9 + 5x^8 + 3x^3$   
 $S(x) = 10x^5 + 7x^4 + 9x^3 + 8x^2 + 2x + 9$ .

$i = 0, \ell = 0$

$$\begin{aligned} r^N(x) &= 10x^6 + 7x^5 + 9x^4 + 8x^3 + 2x^2 + 9x \\ r^0(x) &= x \\ b^N(x) &= 1 \\ b^0(x) &= 0 \\ a^N(x) &= 0 \\ a^0(x) &= 10x \end{aligned}$$

$i = 1, \ell = 0$ :  $d = r_1^N / r_1^0 = 9/1 = 9$

$$\begin{aligned} r^N(x) &= 10x^6 + 7x^5 + 9x^4 + 8x^3 + 2x^2 \\ r^0(x) &= 10x^7 + 7x^6 + 9x^5 + 8x^4 + 2x^3 + 9x^2 \\ b^N(x) &= 1 \\ b^0(x) &= x \\ a^N(x) &= 9x \\ a^0(x) &= 0 \end{aligned}$$



INPUT: SYNDROME POLYNOMIAL  $S(x)$ , INTEGER  $i$

OUTPUT:  $\Lambda(x) = b^N(x)$ ,  $\Omega(x) = a^N(x)/x$

Program 8. CITRON'S ALGORITHM

$$i = 2, \ell = 1: d = r_2^N / r_2^0 = 2/9 = 10$$

$$r^N(x) = 10x^7 + 6x^6 + 5x^5 + 6x^4 + 10x^3$$

$$r^0(x) = 10x^8 + 7x^7 + 9x^6 + 8x^5 + 2x^4 + 9x^3$$

$$b^N(x) = x + 1$$

$$b^0(x) = x^2$$

$$a^N(x) = 9x$$

$$a^0(x) = 0$$

$$i = 3, \ell = 1: d = r_3^N / r_3^0 = 10/9 = 6$$

$$r^N(x) = 6x^8 + x^7 + 7x^6 + x^5 + 5x^4$$

$$r^0(x) = 10x^8 + 6x^7 + 5x^6 + 6x^5 + 10x^4$$

$$b^N(x) = 5x^2 + x + 1$$

$$b^0(x) = x^2 + x$$

$$a^N(x) = 9x$$

$$a^0(x) = 9x^2$$

$$i = 4, \ell = 2: d = r_4^N / r_4^0 = 5/10 = 6$$

$$r^N(x) = x^8 + 9x^7 + 10x^6 + 9x^5$$

$$r^0(x) = 10x^9 + 6x^8 + 5x^7 + 6x^6 + 10x^5$$

$$b^N(x) = 10x^2 + 6x + 1$$

$$b^0(x) = x^3 + x^2$$

$$a^N(x) = x^2 + 9x$$

$$a^0(x) = 9x^3$$

$$i = 5, \ell = 2: d = r_5^N / r_5^0 = 9/10 = 2$$

$$\begin{aligned} r^N(x) &= 2x^9 + 10x^7 + 9x^6 \\ r^0(x) &= x^9 + 9x^8 + 10x^7 + 9x^6 \\ b^N(x) &= 9x^3 + 8x^2 + 6x + 1 \\ b^0(x) &= 10x^3 + 6x^2 + x \\ a^N(x) &= 4x^3 + x^2 + 9x \\ a^0(x) &= x^3 + 9x^2 \end{aligned}$$

$$i = 6, \ell = 3: d = r_6^N / r_6^0 = 9/9 = 1$$

$$\begin{aligned} r^N(x) &= x^9 + 2x^8 \\ r^0(x) &= x^{10} + 9x^9 + 10x^8 + 9x^7 \\ b^N(x) &= 10x^3 + 2x^2 + 5x + 1 \\ b^0(x) &= 10x^4 + 6x^3 + x^2 \\ a^N(x) &= 3x^3 + 3x^2 + 9x \\ a^0(x) &= x^4 + 9x^3 \end{aligned}$$

$i = 7$ , stop.

$$\begin{aligned} \Lambda(x) &= b^N(x) = 10x^3 + 2x^2 + 5x + 1 \\ \Omega(x) &= a^N(x)/x = 3x^2 + 3x + 9 \\ x^{2t}\Lambda(x) &= r^N(x)/x = x^8 + 2x^7 \\ \Lambda(x) &= x^2 + 2x \end{aligned}$$

Program 8 is more efficient than program 7, requiring on the order of  $4t^2$  multiplications instead of  $6t^2$  for correcting  $t$  errors. The respecifications of  $r^0(x)$ ,  $a^0(x)$ , and  $b^0(x)$  in lines 15-17 of the recursive section have been simplified by deletion of the multiplication by  $d^{-1}$ . Like program 7, program 8 requires  $2t$  basic

time units, but a basic time unit now includes a finite field division at every iteration (instead of a finite field inversion at alternate iterations) as well as a multiplication and a subtraction. Any way you look at it, however, this is still the Berlekamp-Massey algorithm. Citron's achievement has been, not to derive a new decoding algorithm, but to show an equivalence between the Berlekamp-Massey algorithm and Kung's generalization of Lanczos' algorithm.

### 7.3 INSIDE EUCLID'S ALGORITHM

In this section we revisit Euclid's algorithm for polynomials (program 3) in order to take apart the polynomial division defined in the first step of the recursion

$$q(x) \leftarrow \left\lfloor r^0(x)/r^N(x) \right\rfloor \quad (84)$$

To keep things as simple as possible, we shall work with the original version of Euclid's algorithm rather than with the extended version which obtains  $a(x)$  and  $b(x)$ . We wish to replace the polynomial division of (84) by a sequence of  $k + 1$  partial divisions where  $k$  is an integer defined by

$$k = \deg(r^0(x)) - \deg(r^N(x)) = \deg(q(x)) \quad (85)$$

Except at the initial iteration, where  $r^0(x) = f(x)$  and  $r^N(x) = g(x)$  may result in  $k = 0$ , we have  $k > 0$  for all polynomial divisions (84). Usually after the first iteration of program 3, though not always,  $q(x)$  is linear so that  $k = 1$ .



In performing the sequence of  $k + 1$  partial divisions we do not want to redefine the divisor  $r^N(x)$  until the polynomial division, i.e., the  $k + 1$ st partial division, is completed. Until that time, the new remainder, say  $r^T(x)$ , becomes the next numerator  $r^0(x)$  and the divisor  $r^N(x)$  is unaltered (except for shifting right to line up appropriately with  $r^0(x)$ ). On the other hand, when the polynomial division is complete, the new remainder becomes the next divisor, while the old divisor becomes the next numerator. Thus, we have

$$\left. \begin{array}{l} r^N(x) \leftarrow x^{-1}r^N(x) \\ r^0(x) \leftarrow r^0(x) - qr^N(x) \end{array} \right\} \begin{array}{l} \text{if not completing} \\ \text{a polynomial division} \end{array} \left. \vphantom{\begin{array}{l} r^N(x) \leftarrow x^{-1}r^N(x) \\ r^0(x) \leftarrow r^0(x) - qr^N(x) \end{array}} \right\} k \text{ times (86)}$$

and

$$\left. \begin{array}{l} r^N(x) \leftarrow r^0(x) - qr^N(x) \\ r^0(x) \leftarrow x^{-1}r^N(x) \end{array} \right\} \begin{array}{l} \text{if completing} \\ \text{a polynomial division} \end{array} \left. \vphantom{\begin{array}{l} r^N(x) \leftarrow r^0(x) - qr^N(x) \\ r^0(x) \leftarrow x^{-1}r^N(x) \end{array}} \right\} k + 1 \text{st (87)}$$

where  $q$  is a scalar defined as the ratio of the leading coefficient of  $r^0(x)$  to the leading coefficient of  $r^N(x)$ .

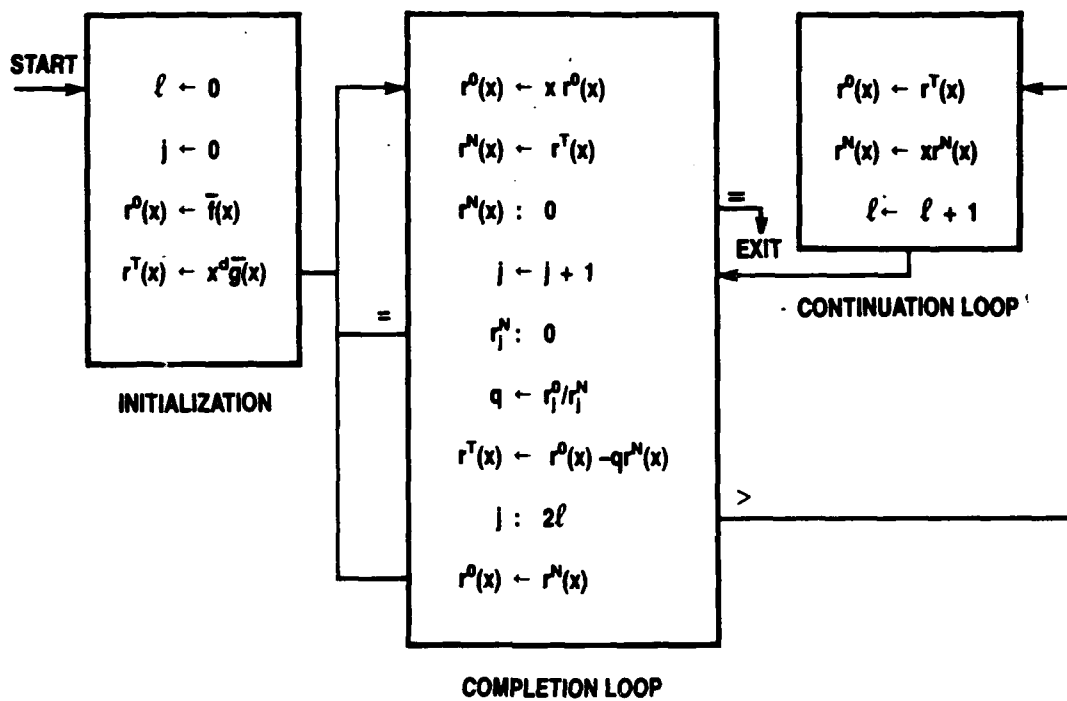
In order to perform the subtractions in (86) and (87) it is necessary to align the leading coefficients of the polynomials  $r^N(x)$  and  $r^0(x)$ . For convenience, we turn the polynomials around and align the trailing coefficients. In this way we can deal with the coefficients  $r_j^N$  and  $r_j^0$  at the  $j$ th iteration and define  $q$  as  $r_j^0/r_j^N$  if  $r_j^N \neq 0$ . Specifically, we shall initialize

$r^0(x)$  by the reversal  $\bar{f}(x)$  of  $f(x)$ . To properly align  $r^0(x)$  and  $r^N(x)$  as  $f(x)$  and  $g(x)$  are initially aligned, we initialize  $r^N(x)$  by  $x^d \bar{g}(x)$ , where  $d = \deg(f(x)) - \deg(g(x))$  is assumed to be strictly positive.

Program 9 is a representation of Euclid's algorithm for polynomials (in the unextended version) when each polynomial division is replaced by a sequence of  $k + 1$  partial divisions, where  $k$  is defined by (85). The polynomials  $r^0(x)$  and  $r^N(x)$  have been initialized by  $\bar{f}(x)$  and  $x^d \bar{g}(x)$ , respectively. Correspondingly, at termination, the  $\gcd(\bar{f}(x), \bar{g}(x))$  is given by  $r^0(x)$ , so that  $\gcd(f(x), g(x))$  is given by  $\bar{r}^0(x)$ , as shown in section 3.

Program 9 is not essentially different from program 3, but shows explicitly what is implied by (84). The recursion is divided into two loops, the left one for completing a polynomial division, and the right loop for continuation of the division. Choice of which loop to follow is determined by the integer variable  $\ell$ .

Assume that as we complete one polynomial division and initiate its successor we have  $j = 2\ell$ . (Observe that  $j$  can never be less than  $2\ell$ , for  $\ell$  is incremented only when  $j > 2\ell$ , and  $j$  is incremented at the same time; on the other hand, if  $j > 2\ell$ , we stay in the continuation loop.) If  $\deg(r^0(x)) = \deg(r^N(x)) + k$  in program 3, then we want to execute the right-hand loop of program 9  $k$  times, followed by one execution of the completion loop. Before the initial incrementation of  $j$  we have  $r_j^0 = 0$ ,  $r_{j+1}^0 \neq 0$ , and  $r_{j+i}^N = 0$  for  $i = 0, \dots, k-1$  and  $r_{j+k}^N \neq 0$ . After the initial incrementation of  $j$ ,  $r_j^0 \neq 0$ . The upper part of



INPUT: POLYNOMIALS  $\bar{f}(x)$ ,  $\bar{g}(x)$ ; INTEGER  $d = \deg(\bar{f}(x)) - \deg(\bar{g}(x)) > 0$   
 OUTPUT:  $\gcd(\bar{f}(x), \bar{g}(x)) = \gamma \bar{r}^0(x)$

Program 9. EUCLID'S ALGORITHM WITHOUT POLYNOMIAL DIVISION

the completion loop is first executed  $k$  times; each iteration increments  $j$  and shifts  $r^0(x)$  once, so that  $r_j^0$  is unchanged as  $j$  increases. After  $k$  executions,  $r_j^N \neq 0$ , so that we enter the continuation loop with  $j = 2\ell + k$ . In the continuation loop,  $r^N(x)$  is shifted once at each iteration, so that  $r_j^N$  now remains fixed and nonzero. We remain in the continuation loop for  $k$  executions, incrementing both  $j$  and  $\ell$ , after which we return to the completion loop with  $j = 2\ell$ . The final remainder  $r^T(x)$  becomes the divisor in the next polynomial division, while the current divisor  $r^N(x)$  becomes the numerator for the next polynomial division. Termination occurs when the new divisor  $r^N(x) = 0$  after completion of a polynomial division. For an example illustrating program 9 we repeat example 2 from section 2.

$$\begin{array}{l} \text{Example 14: } f(x) = x^5 + 3x^4 + 3x^2 + 5x + 10 \\ \quad \quad \quad g(x) = 2x^2 + 7x + 3 \\ \quad \quad \quad d = 5 - 2 = 3 \end{array} \quad \left. \vphantom{\begin{array}{l} f(x) \\ g(x) \\ d \end{array}} \right\} \text{ over GF(11)}$$

$$j = 0, \ell = 0$$

$$\begin{aligned} r^0(x) \leftarrow x^{\ell} f(x) &= 10x^6 + 5x^5 + 3x^4 + 3x^2 + x \\ r^N(x) \leftarrow x^3 \bar{g}(x) &= 3x^5 + 7x^4 + 2x^3 \end{aligned}$$

$$j = 1, \ell = 0, r_j^N = 0$$

$$r^0(x) \leftarrow x r^0(x) = 10x^7 + 5x^6 + 3x^5 + 3x^3 + x^2$$

$$j = 2, \ell = 0, r_j^N = 0$$

$$r^0(x) \leftarrow x r^0(x) = 10x^8 + 5x^7 + 3x^6 + 3x^4 + x^3$$

$$j = 3, \ell = 0, r_j^N = 2, r_j^0 = 1, q = 1/2 = 6$$

$$\begin{aligned} r^T(x) + r^0(x) - qr^N(x) &= 10x^8 + 5x^7 + 3x^6 + 4x^5 + 5x^4 \\ r^0(x) + r^T(x) \\ r^N(x) + xr^N(x) &= 3x^6 + 7x^5 + 2x^4 \end{aligned}$$

$$j = 4, \ell = 1, r_j^N = 2, r_j^0 = 5, q = 5/2 = 8$$

$$\begin{aligned} r^T(x) + r^0(x) - qr^N(x) &= 10x^8 + 5x^7 + x^6 + 3x^5 \\ r^0(x) + r^T(x) \\ r^N(x) + xr^N(x) &= 3x^7 + 7x^6 + 2x^5 \end{aligned}$$

$$j = 5, \ell = 2, r_j^N = 2, r_j^0 = 3, q = 3/2 = 7$$

$$\begin{aligned} r^T(x) + r^0(x) - qr^N(x) &= 10x^8 + 6x^7 + 7x^6 \\ r^0(x) + r^T(x) \\ r^N(x) + xr^N(x) &= 3x^8 + 7x^7 + 2x^6 \end{aligned}$$

$$j = 6, \ell = 3, r_j^N = 2, r_j^0 = 7, q = 7/2 = 9$$

$$\begin{aligned} r^T(x) + r^0(x) - qr^N(x) &= 5x^8 + 9x^7 \\ r^0(x) + xr^N(x) &= 3x^9 + 7x^8 + 2x^7 \\ r^N(x) + r^T(x) \end{aligned}$$

$$j = 7, \ell = 3, r_j^N = 9, r_j^0 = 2, q = 2/9 = 10$$

$$\begin{aligned} r^T(x) + r^0(x) - qr^N(x) &= 3x^9 + x^8 \\ r^0(x) + r^T(x) \\ r^N(x) + xr^N(x) &= 5x^9 + 9x^8 \end{aligned}$$

$$j = 8, \ell = 4: r_j^N = 9, r_j^0 = 1, q = 1/9 = 5$$

$$r^T(x) \leftarrow r^0(x) - qr^N(x) = 0$$

$$r^0(x) \leftarrow xr^N(x) = 5x^{10} + 9x^9$$

$$r^N(x) \leftarrow r^T(x) = 0$$

stop

$$\beta \cdot \gcd(f(x), g(x)) = \overline{r^0(x)} = 9x + 5$$

$$\beta = 9, \gamma = \beta^{-1} = 5$$

$$\gcd(f(x), g(x)) = x + 3$$

A comparison of this example with example 2 shows that the sequence of  $q$ 's defined above is identical to the successive coefficients of  $q(x)$  obtained in the earlier example.

Program 9 is somewhat awkward because two different loops are followed according as we are concluding a completion loop or a continuation loop. Both loops contain some statements in common, namely, lines 4-8 in the completion box. Both loops define a new polynomial  $r^T(x)$  at line 7, and retain this new polynomial together with one of the pair  $(r^N(x), r^0(x))$ . In the continuation loop  $r^T(x)$  becomes the new numerator, while the other retained polynomial (shifted) becomes the divisor; in the completion loop the assignments are reversed:  $r^T(x)$  becomes the new divisor, while the other retained polynomial (shifted) becomes the numerator. Surprisingly, all that really matters is that the

correct pair of polynomials be retained. It does not matter which polynomial is assigned to be the numerator and which the divisor. We can take advantage of this fact to design an improved algorithm.

Let us first observe that it is permissible to multiply either  $r^0(x)$  or  $r^N(x)$  by an arbitrary scalar  $\beta$ . For if

$$R^0(x) = \beta r^0(x)$$

and

$$R^N(x) = \gamma r^N(x)$$

for some scalars  $\beta$  and  $\gamma$ , then

$$Q = R_j^0 / R_j^N = (\beta/\gamma)q$$

$$\text{and } R^T(x) = R^0(x) - QR^N(x)$$

$$= \beta r^0(x) - (\beta/\gamma)q\gamma r^N(x)$$

$$= \beta r^T(x).$$

Thus the only effect produced by multiplying  $r^0(x)$  or  $r^N(x)$  by a scalar is to multiply future  $r^0(x)$  and  $r^N(x)$  by some scalar.

Next, consider the effect of swapping roles. If the assignments of polynomials to  $r^0(x)$  and  $r^N(x)$  are reversed at some stage, then we shall calculate a new  $R^T(x)$ , say, at line 7 by

$$R^T(x) = r^N(x) - (r_j^N/r_j^0)r^0(x)$$

so that

$$-qr^T(x) = -qr^N(x) + r^0(x) = r^T(x)$$

or

$$R^T(x) = -q^{-1}r^T(x).$$

Thus, reversing the roles of numerator and divisor has no effect on the algorithm other than a multiplication of the result by a scalar so long as we take care to retain the correct pair of polynomials at each iteration.

We are now in a position to eliminate the continuation loop from program 9, producing an improved version of Euclid's algorithm. We shall still replace each polynomial division by a sequence of  $k + 1$  partial divisions. After the initial determination of  $r^T(x)$  (in a given polynomial division) we omit the branch to the continuation loop, define  $r^0(x)$  by the old divisor in the last line, and let  $r^T(x)$  become the divisor at line 2, thus reversing the roles played by  $r^0(x)$  and  $r^N(x)$ .

Having once defined  $r^0(x)$  at line 9 of the completion loop, however, we must not redefine it (except for shifts in line 1) until the polynomial division is completed. This we can ensure by adding the statement



$$\ell \leftarrow j - \ell$$

to the end of the completion box, and replacing the branch from the test

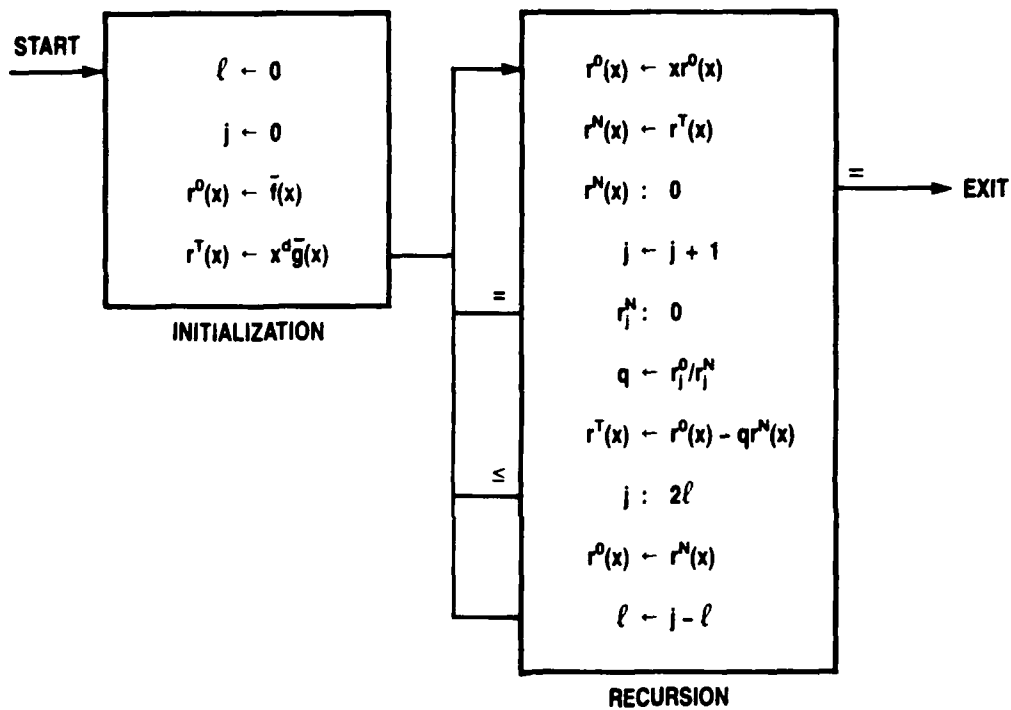
$$j : 2\ell$$

with a branch to the first line whenever the relation is satisfied by  $\leq$ . The first time through we have  $j = 2\ell + k$ . At the new last line of the box,  $\ell$  is redefined as  $\ell' = j - \ell = \ell + k$ . Thereafter, the branch test will succeed for the next  $k$  iterations, until  $j = 2\ell + 2k = 2\ell'$ , when the polynomial division is finally completed. During these  $k$  iterations the roles of  $r^0(x)$  and  $r^N(x)$  remain reversed. The divisor polynomials of program 9 are now numerators, and scalar multiples of the numerators of program 9 are now divisors.

There is one more point to be made. During the  $k$  iterations with reversed roles  $r_j^0$  is fixed and nonzero (as  $r_j^N$  was fixed and nonzero in program 9). However, it is possible that at some one of these iterations  $r_j^N$  is zero, causing a branch to line 1 from line 5. This is more efficient than the longer path taken in program 9, where  $q$  is defined as 0,  $r^T(x)$  as  $r^0(x)$ , followed by a branch to the continuation loop which redefines  $r^0(x)$  as  $r^T(x)$ , i.e. as itself. The result is the same; the path taken is longer in program 9.

Our final version of Euclid's algorithm is given by program 10 and illustrated by example 15, a reworking of examples 2 and 14.

$$\text{Example 15: } \left. \begin{array}{l} f(x) = x^5 + 3x^4 + 3x^2 + 5x + 10 \\ g(x) = 2x^2 + 7x + 3 \\ d = 3 \end{array} \right\} \text{ over GF(11)}$$



INPUT: POLYNOMIALS  $\bar{f}(x), \bar{g}(x)$ ; INTEGER  $d = \deg(f(x)) - \deg(g(x)) > 0$

OUTPUT:  $\gcd(f(x), g(x)) = \gamma \bar{r}^0(x)$

Program 10. SIMPLIFIED EUCLID'S ALGORITHM

$$j = 0, \ell = 0$$

$$\begin{aligned} r^0(x) + x\bar{f}(x) &= 10x^6 + 5x^5 + 3x^4 + 3x^2 + x \\ r^N(x) + x^3\bar{g}(x) &= 3x^5 + 7x^4 + 2x^3 \end{aligned}$$

$$j = 1, \ell = 0: r_j^N = 0,$$

$$r^0(x) + xr^0(x) = 10x^7 + 5x^6 + 3x^5 + 3x^3 + x^2$$

$$j = 2, \ell = 0, r_j^N = 0,$$

$$r^0(x) + xr^0(x) = 10x^8 + 5x^7 + 3x^6 + 3x^4 + x^3$$

$$j = 3, \ell = 0: r_j^N = 2, r_j^0 = 1, q = 1/2 = 6$$

$$\begin{aligned} r^T(x) + r^0(x) - qr^N(x) &= 10x^8 + 5x^7 + 3x^6 + 4x^5 + 5x^4 \\ r^0(x) + xr^N(x) &= 3x^6 + 7x^5 + 2x^4 \\ r^N(x) + r^T(x) \end{aligned}$$

$$j = 4, \ell = 3: r_j^N = 5, r_j^0 = 2, q = 2/5 = 7$$

$$\begin{aligned} r^T(x) + r^0(x) - qr^N(x) &= 7x^8 + 9x^7 + 4x^6 + x^5 \\ r^0(x) + xr^0(x) &= 3x^7 + 7x^6 + 2x^5 \\ r^N(x) + r^T(x) \end{aligned}$$

$$j = 5, \ell = 3: r_j^N = 1, r_j^0 = 2, q = 2/1 = 2$$

$$\begin{aligned} r^T(x) + r^0(x) - qr^N(x) &= 8x^8 + 7x^7 + 10x^6 \\ r^0(x) + xr^0(x) &= 3x^8 + 7x^7 + 2x^6 \\ r^N(x) + r^T(x) \end{aligned}$$

$$j = 6, \ell = 3: r_j^N = 10, r_j^0 = 2, q = 2/10 = 9$$

$$\begin{aligned} r^T(x) + r^0(x) - qr^N(x) &= 8x^8 + 10x^7 \\ r^0(x) + xr^0(x) &= 3x^9 + 7x^8 + 2x^7 \\ r^N(x) + r^T(x) \end{aligned}$$

$$j = 7, \ell = 3: r_j^N = 10, r_j^0 = 2, q = 2/10 = 9$$

$$r^T(x) \leftarrow r^0(x) - qr^N(x) = 3x^9 + x^8$$

$$r^0(x) \leftarrow xr^N(x) = 8x^9 + 10x^8$$

$$r^N(x) \leftarrow r^T(x)$$

$$j = 8, \ell = 4: r_j^N = 1, r_j^0 = 10, q = 10/1 = 10$$

$$r^T(x) \leftarrow r^0(x) - qr^N(x) = 0$$

$$r^0(x) \leftarrow xr^N(x) = 8x^{10} + 10x^9$$

$$r^N(x) \leftarrow r^T(x) = 0$$

stop

$$\beta = \gcd(f(x), g(x)) = \overline{r^0(x)} = 10x + 8$$

$$\beta = 10, \gamma = \beta^{-1} = 10$$

$$\gcd(f(x), g(x)) = x + 3$$

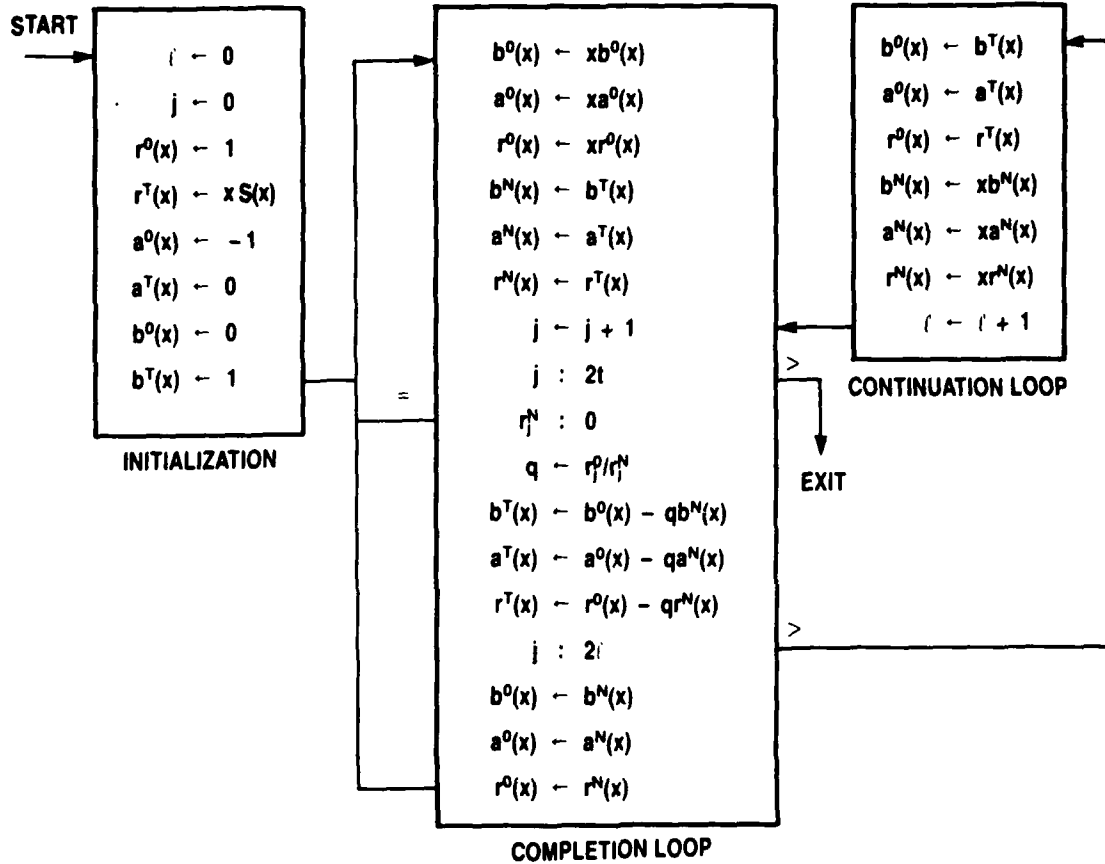
Program 9 is an exact translation of Euclid's algorithm for polynomials when polynomial division is broken down. Program 10 is cleaner and more efficient than program 9. Program 10 closely parallels Berlekamp's decoding algorithm and, in effect, shows why the Berlekamp-Massey algorithm is more efficient than decoding algorithms based directly on Euclid's algorithm. In section 7.4 we adapt the Mills' decoding algorithm of program 5 to reflect the changes of programs 9 and 10. The resulting decoding algorithms are then shown to be equivalent to the Euclideanized Berlekamp-Massey algorithm of program 8.

#### 7.4 MILLS' ALGORITHM IN BERLEKAMP-MASSEY DRESS

In this section the Mills' decoding algorithm of program 5 is modified in two stages. First, the polynomial division is replaced by a sequence of partial divisions as in program 9. The resulting algorithm is essentially the same as program 5, but is free of polynomial divisions and can test for termination by counting iterations. However, like program 9 it suffers from a more complicated control structure in that the recursive section consists of two distinct loops. In the second stage we eliminate the continuation loop, producing an algorithm analogous to program 10. This version of Mills' algorithm closely parallels program 8 and might be viewed as its Euclidean reflection. Finally, we show that these new decoding algorithms are equivalent to the Berlekamp-Massey algorithm of program 8.

An initial change which we make in program 5 in order to conform to the initializations of programs 7 and 8 is to reverse the signs of the initial values of  $r^0(x)$  and  $a^0(x)$ . In program 5, this would have the effect of reversing the sign of  $q(x)$  at each iteration and of  $r(x)$ ,  $a(x)$ , and  $b(x)$  at each odd-numbered iteration. Since at termination  $\bar{\Lambda}(x)$  is obtained as some scalar multiple of  $b^N(x)$ , and  $\bar{Q}(x)$  as the same multiple of  $a^N(x)$ , this sign reversal may change the scalar but does not affect the determination of  $\Lambda(x)$  and  $Q(x)$ , nor of the error magnitudes.

As in programs 9 and 10, it is convenient to be able to define  $q$  at the  $j$ th iteration as  $r_j^0/r_j^N$  instead of as the ratio of the leading coefficients. To achieve this, we initialize  $r^N(x)$  by  $xS(x)$  and  $r^0(x)$  by 1, as in programs 7 and 8, rather than by  $\bar{S}(x)$  and  $-x^{2t}$ , as in program 5. Initialization of  $r^N(x)$  by  $xS(x)$



INPUT: SYNDROME POLYNOMIAL  $S(x)$ , INTEGER  $t$

OUTPUT:  $\gamma \wedge(x) = b^N(x)$ ,  $\gamma \Omega(x) = a^N(x)/x$

Program 11. MILLS' ALGORITHM WITH PARTIAL DIVISIONS

means that at iteration 1,  $r_1^N$  will be  $S_1$ ; initialization of  $r^0(x)$  by 1 means that at iteration 1, after a right shift of  $r^0(x)$ ,  $r_1^0$  will be 1, and the initial  $q$  is defined as  $1/S_1$ , as required, 1 and  $S_1$  being the leading coefficients of  $x^{2t}$  and  $\tilde{S}(x)$ .

Program 11 is a representation of Mills' algorithm with these initialization changes when the polynomial division is broken down into its partial divisions (i.e., program 11 is the Mills' decoder analog of program 9). This is not a different algorithm from that represented in program 5, but explicitly shows what is implied by the first statement in the recursion of program 5

$$q(x) \leftarrow \lfloor r^0(x)/r^N(x) \rfloor.$$

The recursion in program 11 is divided into two loops, the left one for completing a polynomial division, and the right loop for continuation of the division as in program 9. Choice of which loop to follow is determined by the integer variable  $\lambda$ .

Termination of the program can now be decided by counting iterations  $j$  and stopping if  $j$  exceeds  $2t$ . For, if in program 5  $\deg(r^N(x)) < t$  then in program 11  $r_j^N = 0$  and the program makes no further changes except to increment  $j$ . Suppose  $2t$  iterations do not suffice. Each polynomial division with  $k = \deg(q(x))$  requires  $2k$  iterations ( $k$  shifts of  $r^0(x)$  followed by  $k$  trips through the continuation loop). Thus, if  $n$  polynomial divisions are required, and  $k_i$  denotes the degree of the  $i$ th quotient polynomial as defined by (85) for the remainder polynomials of program 5, then

$$2 \sum_{i=1}^n k_i = 2t + 2s$$

where  $s > 0$  if  $2t$  iterations do not suffice.

But

$$\begin{aligned} \sum_{i=1}^n k_i &= \deg(r^{-1}(x)) - \deg(r^{n-1}(x)) \\ &= 2t - \deg(r^{n-1}(x)) \end{aligned}$$

Therefore,

$$\begin{aligned} \deg(r^{n-1}(x)) &= 2t - (t + s) \\ &= t - s < t \end{aligned}$$

leading to a contradiction. Therefore,  $2t$  iterations suffice and program 11 need not test for the degree of  $r^N(x)$ .

The treatment of  $r(x)$  is slightly different from that of program 5. We need to keep only  $2t$  terms, and at each iteration  $j$  we set  $r_j^T = 0$ , leaving only  $2t - j$  coefficients to be multiplied at the next update. We thus require only  $2t^2 + t$  multiplications



for updating  $r^T(x)$ , instead of the  $3t^2 + t$  implied by program 5. In program 5, the number of multiplications could also be reduced from  $3t^2 + t$  to  $2t^2 + t$  by recognizing that terms in  $r(x)$  beyond  $2t - j$  need not be retained after iteration  $j$ . However, this same reduction cannot be applied in program 4 without losing  $Q(x)$  in the process.  $2t$  basic time units are required in program 11 to correct  $t$  errors, where a basic time unit consists of the time required for one finite field division, one multiplication, and one subtraction.

We now repeat our Reed-Solomon 3-error correcting code example for program 11.

Example 16: Reed-Solomon 3-error-correcting code over  $GF(11)$  with  $\alpha = 2$ .

$t = 3$ ; let  $c(x) = 0$ ,  $v(x) = e(x) = 6x^9 + 5x^8 + 3x^3$ .  
 $S(x) = 10x^5 + 7x^4 + 9x^3 + 8x^2 + 2x + 9$

$j = 0$ ,  $v = 0$ :

$$\begin{aligned} r^N(x) &= xS(x) = 10x^6 + 7x^5 + 9x^4 + 8x^3 + 2x^2 + 9x \\ r^0(x) &= x \text{ (after shift)} \\ b^N(x) &= 1 \\ b^0(x) &= 0 \\ a^N(x) &= 0 \\ a^0(x) &= -x = 10x \end{aligned}$$

$j = 1$ ,  $\ell = 0$ :  $q = r_1^0/r_1^N = 1/9 = 5$ ; execute continuation loop

$$\begin{aligned} r^N(x) &\leftarrow xr^N(x) = 10x^7 + 7x^6 + 9x^5 + 8x^4 + 2x^3 + 9x^2 \\ r^0(x) &\leftarrow r^0(x) - qr^N(x) = 5x^6 + 9x^5 + 10x^4 + 4x^3 + x^2 \\ b^N(x) &\leftarrow xb^N(x) = x \\ b^0(x) &\leftarrow b^0(x) - qb^N(x) = 6 \\ a^N(x) &\leftarrow xa^N(x) = 0 \\ a^0(x) &\leftarrow a^0(x) - qa^N(x) = 10x \end{aligned}$$

$j = 2, \lambda = 1: q = r_2^0/r_2^N = 1/9 = 5; \text{ execute completion loop}$

$$r^N(x) \leftarrow r^0(x) - qr^N(x) = 5x^7 + 3x^6 + 8x^5 + 3x^4 + 5x^3$$

$$r^0(x) \leftarrow xr^N(x) = 10x^8 + 7x^7 + 9x^6 + 8x^5 + 2x^4 + 9x^3$$

$$b^N(x) \leftarrow b^0(x) - qb^N(x) = 6x + 6$$

$$b^0(x) \leftarrow xb^N(x) = x^2$$

$$a^N(x) \leftarrow a^0(x) - qa^N(x) = 10x$$

$$a^0(x) \leftarrow xa^N(x) = 0$$

$j = 3, \lambda = 1: q = r_3^0/r_3^N = 9/5 = 4; \text{ execute continuation loop}$

$$r^N(x) = 5x^8 + 3x^7 + 8x^6 + 3x^5 + 5x^4$$

$$r^0(x) = 10x^9 + 9x^7 + 8x^6 + 9x^5 + x^4$$

$$b^N(x) = 6x^2 + 6x$$

$$b^0(x) = x^2 + 9x + 9$$

$$a^N(x) = 10x^2$$

$$a^0(x) = 4x$$

$j = 4, \lambda = 2: q = r_4^0/r_4^N = 1/5 = 9; \text{ execute completion loop}$

$$r^N(x) = 9x^8 + 4x^7 + 2x^6 + 4x^5$$

$$r^0(x) = 5x^9 + 3x^8 + 8x^7 + 3x^6 + 5x^5$$

$$b^N(x) = 2x^2 + 10x + 9$$

$$b^0(x) = 6x^3 + 6x^2$$

$$a^N(x) = 9x^2 + 4x$$

$$a^0(x) = 10x^3$$

$j = 5, \lambda = 2: q = r_5^0/r_5^N = 5/4 = 4; \text{ execute continuation loop}$

$$r^N(x) = 9x^9 + 4x^8 + 2x^7 + 4x^6$$

$$r^0(x) = 5x^9 + 3x^7 + 6x^6$$

$$b^N(x) = 2x^3 + 10x^2 + 9x$$

$$b^0(x) = 6x^3 + 9x^2 + 4x + 8$$

$$a^N(x) = 9x^3 + 4x^2$$

$$a^0(x) = 10x^3 + 8x^2 + 6x$$

$j = 6, \ell = 3: q = r_6^0/r_6^N = 6/4 = 7$ ; execute completion loop

$$\begin{aligned} r^N(x) &= 8x^9 + 5x^8 \\ r^0(x) &= 9x^{10} + 4x^9 + 2x^8 + 4x^7 \\ b^N(x) &= 3x^3 + 5x^2 + 7x + 8 \\ b^0(x) &= 2x^4 + 10x^3 + 9x^2 \\ a^N(x) &= 2x^3 + 2x^2 + 6x \\ a^0(x) &= 9x^4 + 4x^3 \end{aligned}$$

$j = 7$ , stop:  $\gamma = 8, \gamma^{-1} = 7$

$$\begin{aligned} \Lambda(x) &= \gamma^{-1} r^N(x) = 10x^3 + 2x^2 + 5x + 1 \\ \cap(x) &= \gamma^{-1} a^N(x)/x = 3x^2 + 3x + 9 \\ x^2 t_1(x) &= \gamma^{-1} r^N(x)/x = x^8 + 2x^7 \\ \lambda(x) &= x^2 + 2x \end{aligned}$$

The computational flow of program 11 is awkward, like that of program 9. A more efficient algorithm can be obtained by incorporating the changes of program 10 into Mills' decoding algorithm. Three alterations are made to program 11:

- (1) Eliminate the right-hand loop, returning the arrow to the top of the left-hand loop.
- (2) Reverse the condition for taking the branch from ">" to "≤".
- (3) Add the specification statement

$$\ell \leftarrow j - \ell$$

to the bottom of the left-hand loop.

These changes result in program 12, which is seen to be very similar to program 8. Like programs 8 and 9, program 12 requires  $2t^2 + t$  multiplications for updating  $r^T(x)$ , and a total on the order of  $4t^2$  multiplications for finding  $\Lambda(x)$  when  $t$  errors have occurred. Like programs 8 and 9, program 12 requires  $2t$  basic time units where a basic time unit includes the time to perform one finite field division, one multiplication, and one subtraction.

We now repeat example 16 for program 12.

Example 17: Reed-Solomon 3-error-correcting code over  $GF(11)$  with  $\alpha = 2$ .

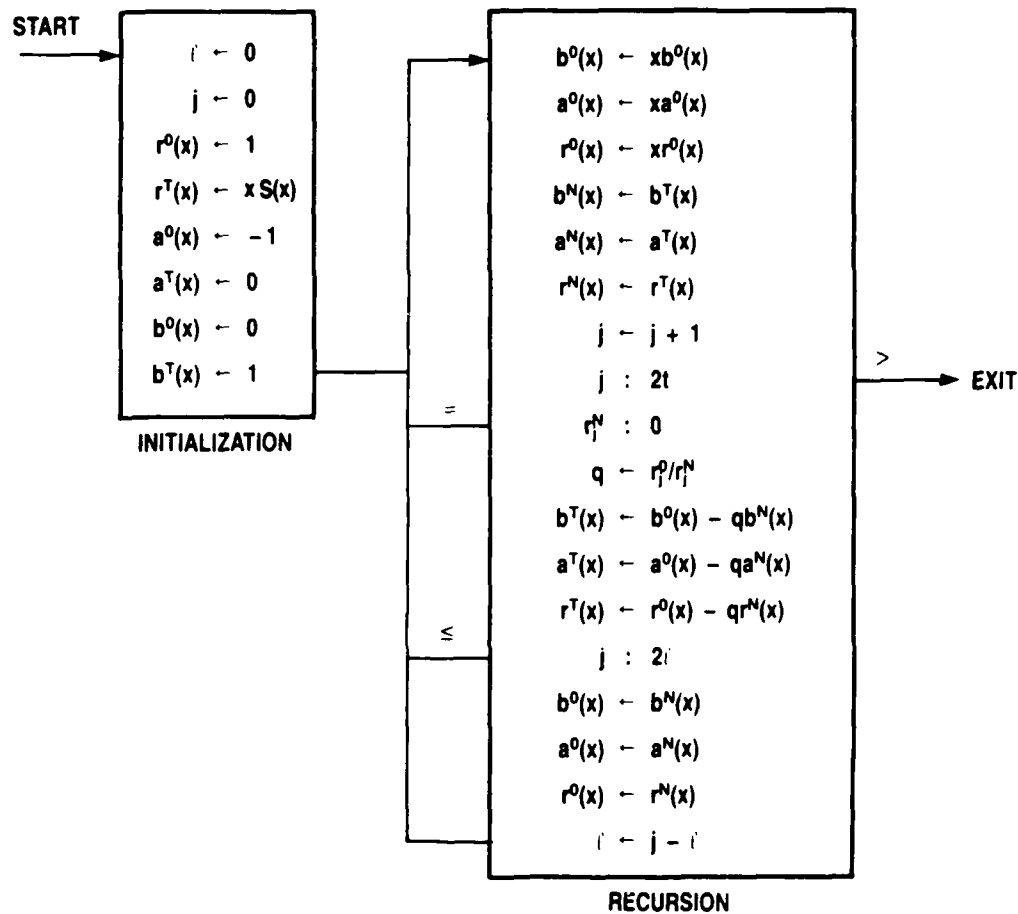
$t = 3$ ; let  $c(x) = 0$ ,  $v(x) = e(x) = 6x^9 + 5x^8 + 3x^3$ .  
 $S(x) = 10x^5 + 7x^4 + 9x^3 + 8x^2 + 2x + 9$

$j = 0, \ell = 0$ :

$$\begin{aligned} r^N(x) &= xS(x) = 10x^6 + 7x^5 + 9x^4 + 8x^3 + 2x^2 + 9x \\ r^0(x) &= x \\ b^N(x) &= 1 \\ b^0(x) &= 0 \\ a^N(x) &= 0 \\ a^0(x) &= -x = 10x \end{aligned}$$

$j = 1, \ell = 0$ :  $q = r_1^0 / r_1^N = 1/9 = 5$

$$\begin{aligned} r^N(x) &= 5x^6 + 9x^5 + 10x^4 + 4x^3 + x^2 \\ r^0(x) &= 10x^7 + 7x^6 + 9x^5 + 8x^4 + 2x^3 + 9x^2 \\ b^N(x) &= 6 \\ b^0(x) &= x \\ a^N(x) &= 10 \\ a^0(x) &= 0 \end{aligned}$$



INPUT: SYNDROME POLYNOMIAL  $S(x)$ , INTEGER  $t$

OUTPUT:  $\gamma \wedge(x) = b^N(x)$ ,  $\gamma \Omega(x) = a^N(x)/x$

Program 12. SIMPLIFIED MILLS' ALGORITHM

$$j = 2, \ell = 1: q = r_2^0/r_2^N = 9/1 = 9$$

$$\begin{aligned} r^N(x) &= 10x^7 + 6x^6 + 5x^5 + 6x^4 + 10x^3 \\ r^0(x) &= 10x^8 + 7x^7 + 9x^6 + 8x^5 + 2x^4 + 9x^3 \\ b^N(x) &= x + 1 \\ b^0(x) &= x^2 \\ a^N(x) &= 9x \\ a^0(x) &= 0 \end{aligned}$$

$$j = 3, \ell = 1: q = r_3^0/r_3^N = 9/10 = 2;$$

$$\begin{aligned} r^N(x) &= 10x^8 + 9x^7 + 8x^6 + 9x^5 + x^4 \\ r^0(x) &= 10x^8 + 6x^7 + 5x^6 + 6x^5 + 10x^4 \\ b^N(x) &= x^2 + 9x + 9 \\ b^0(x) &= x^2 + x \\ a^N(x) &= 4x \\ a^0(x) &= 9x^2 \end{aligned}$$

$$j = 4, \ell = 2: q = r_4^0/r_4^N = 10/1 = 10$$

$$\begin{aligned} r^N(x) &= 9x^8 + 4x^7 + 2x^6 + 4x^5 \\ r^0(x) &= 10x^9 + 6x^8 + 5x^7 + 6x^6 + 10x^5 \\ b^N(x) &= 2x^2 + 10x + 9 \\ b^0(x) &= x^3 + x^2 \\ a^N(x) &= 9x^2 + 4x \\ a^0(x) &= 9x^3 \end{aligned}$$

$$j = 5, \ell = 2: q = r_5^0/r_5^N = 10/4 = 8$$

$$\begin{aligned} r^N(x) &= 10x^9 + 0x^8 + 6x^7 + x^6 \\ r^0(x) &= 9x^9 + 4x^8 + 2x^7 + 4x^6 \\ b^N(x) &= x^3 + 7x^2 + 8x + 5 \\ b^0(x) &= 2x^3 + 10x^2 + 9x \\ a^N(x) &= 9x^3 + 5x^2 + x \\ a^0(x) &= 9x^3 + 4x^2 \end{aligned}$$

$$j = 6, \ell = 3: q = r_6^0/r_6^N = 4/1 = 4$$

$$\begin{aligned} r^N(x) &= 2x^9 + 4x^8 \\ r^0(x) &= 9x^{10} + 4x^9 + 2x^8 + 4x^7 \\ b^N(x) &= 9x^3 + 4x^2 + 10x + 2 \\ b^0(x) &= 2x^4 + 10x^3 + 9x^2 \\ a^N(x) &= 6x^3 + 6x^2 + 7x \\ a^0(x) &= 9x^4 + 4x^3 \end{aligned}$$

$$j = 7, \text{ stop: } \gamma = 2, \gamma^{-1} = 6$$

$$\begin{aligned} \Lambda(x) &= \gamma^{-1}r^N(x) = 10x^3 + 2x^2 + 5x + 1 \\ \cap(x) &= \gamma^{-1}a^N(x)/x = 3x^2 + 3x + 9 \\ x^2t^N(x) &= \gamma^{-1}r^N(x)/x = x^8 + 2x^7 \\ \lambda(x) &= x^2 + 2x \end{aligned}$$

Transformation of program 11 into program 12 is justified by the same arguments used for validating the transformation of program 9 into program 10. We now demonstrate that at each iteration  $j$  the polynomials  $a^j(x)$ ,  $b^j(x)$ , and  $r^j(x)$  produced by program 12 differ from those produced by program 8 only by a scale factor. To distinguish between the polynomials produced by the two algorithms we shall use lower case  $r(x)$ , etc., for program 12 (Mills) and upper case  $R(x)$ , etc., for program 8 (Berlekamp-Massey). We assume that at iteration  $j$  at the instant when the counter  $j$  is incremented the polynomials  $r^0(x)$  and  $r^N(x)$  are related to  $R^0(x)$  and  $R^N(x)$  by

$$r^0(x) = \beta R^0(x)$$

$$r^N(x) = \gamma R^N(x)$$

where  $\alpha$  and  $\gamma$  are scalars (field elements), and show that if  $r_j^N \neq 0$

$$r^T(x) = -q\gamma R^T(x)$$

where

$$q = r_j^0/r_j^N = (\beta/\gamma)d^{-1}$$

where  $d$  is the quantity computed at line 9 in the recursion section of program 8. We have

$$R^T(x) = R^N(x) - dR^0(x)$$

or

$$-d^{-1}R^T(x) = R^0(x) - d^{-1}R^N(x).$$

Therefore,

$$-(\gamma/\beta)qR^T(x) = \alpha^{-1}r^0(x) - (\gamma/\beta)q\gamma^{-1}r^N(x).$$

Multiplying by  $\beta$ , we find that

$$-q\gamma R^T(x) = r^0(x) - qr^N(x) = r^T(x).$$

Thus, each time  $r_j^N \neq 0$ , the ratios  $r^N(x)/R^N(x)$  are multiplied by  $-q$  to obtain  $r^T(x)/R^T(x)$ ; when  $r_j^N = 0$ ,  $R_j^N = 0$  so that  $q$



is undefined and  $r^T(x)$  and  $R^T(x)$  are unchanged. For  $j = 0$ ,  $r_j^N = R_j^N$ , so that at iteration  $j$

$$r^T(x) = \prod_{\substack{i=1 \\ q_i \text{ defined}}}^j (-q_i) R^T(x). \quad (88)$$

The same relationship holds between  $a^T(x)$  and  $A^T(x)$  and between  $b^T(x)$  and  $B^T(x)$ . Observe that  $q_j$  is never zero in program 12, because  $r^0(x)$  is initialized as 1,  $r^0(x)$  is redefined as  $r^N(x)$  only when  $r_j^N \neq 0$ , and  $r_j^0$  does not change when  $r_j^N = 0$ . Since program 8 solves the key equation, it follows that program 12 must solve the key equation (14) for  $\Lambda(x)$  and  $\Omega(x)$ . Again, we do not care if the polynomials differ from those produced by program 8 by a scale factor, for neither the Chien search nor application of Forney's formula (16) is thereby affected.

Let

$$M_j = \prod_{\substack{i=1 \\ q_i \text{ defined}}}^j (-q_i)$$

in expression (88). Table 1 shows the values of  $M_j$ ,  $r^N(x)$ , and  $R^N(x)$  obtained in a comparison of examples 17 and 13. From this table it is easily seen that equation (88) is satisfied by the outputs for  $r^N(x)$  and  $R^N(x)$  from programs 8 and 12 for this example. Similar tables can be constructed for  $a^N(x)$  and  $b^N(x)$ .

Table 1. A Comparison of Outputs from Programs 12 ( $r^N(x)$ ) and 8 ( $R^N(x)$ )

j	qj	$r^N(x)$	Mj	$R^N(x)$
1	5	$5x^6+9x^5+10x^4+4x^3+x^2$	6	$10x^6+7x^5+9x^4+8x^3+2x^2$
2	9	$10x^7+6x^6+5x^5+6x^4+10x^3$	1	$10x^7+6x^6+5x^5+6x^4+10x^3$
3	2	$10x^8+9x^7+8x^6+9x^5+x^4$	9	$6x^8+x^7+7x^6+x^5+5x^4$
4	10	$9x^8+4x^7+2x^6+4x^5$	9	$x^8+9x^7+10x^6+9x^5$
5	8	$10x^9+6x^7+x^6$	5	$2x^9+10x^7+9x^6$
6	4	$2x^9+4x^8$	2	$x^9+2x^8$

We now go back and show that at each iteration  $j$  the polynomials  $a^j(x)$ ,  $b^j(x)$ , and  $r^j(x)$  produced by program 11 differ from those produced by program 12, and hence from those produced by program 8, only by a scale factor. Since program 11 is equivalent to program 5, this demonstrates the equivalence of Mills' algorithm and Berlekamp's algorithm.

To distinguish polynomials produced by program 11 from those produced by program 12 we shall use lower case  $r(x)$ , etc., for program 11 and upper case  $R(x)$ , etc., for program 12. We now have two separate cases to treat, according as we have just executed the completion loop or the continuation loop in program 11. In the former case the branch at line 14 of the recursion of program 12 is not taken; in the latter case the branch is always taken.

#### Case 1: Completion Loop

We assume

$$r^0(x) = \beta R^0(x)$$

$$r^N(x) = \gamma R^N(x)$$

for some field elements  $\beta$  and  $\gamma$ , and show that

$$r^T(x) = \delta R^T(x)$$

for some field element  $\delta$ , namely  $\beta$ .

We have

$$q = r_j^0 / r_j^N = \beta R_j^0 / \gamma R_j^N = (\beta / \gamma) Q$$

$$\begin{aligned} r^T(x) &= r^0(x) - qr^N(x) \\ &= \beta R^0(x) - (\beta / \gamma) Q \gamma R^N(x) \\ &= \beta R^T(x) \end{aligned} \tag{89}$$

so that  $\delta = \beta$ , as claimed. At the conclusion of the iteration, we then have

$$r^0(x) \text{ (new)} = r^T(x) = \beta R^T(x) = \beta R^N(x) \text{ (new)}$$

and

$$r^N(x) \text{ (new)} = \gamma r^N(x) \text{ (old)} = \gamma x R^N(x) \text{ (old)} = \gamma R^0(x) \text{ (new)},$$

where  $r^0(x) \text{ (new)}$  denotes the value of  $r^0(x)$  just prior to the next incrementation of the counter  $j$ ,  $r^N(x) \text{ (old)}$  denotes the value of  $r^N(x)$  at the last incrementation of the counter  $j$ , etc.

## Case 2: Continuation

We assume

$$\begin{aligned}r^0(x) &= \beta R^N(x) \\ r^N(x) &= \gamma R^0(x)\end{aligned}$$

for some field elements  $\beta$  and  $\gamma$ , and show that

$$r^T(x) = \delta R^T(x)$$

for some field element  $\delta$ , namely  $-\beta/Q$ .

We have

$$q = r_j^0/r_j^N = \beta R_j^N/\gamma R_j^0 = (\beta/\gamma)Q^{-1}$$

(Recall that  $Q \neq 0$ , for  $q$  is never 0 in program 10.)

$$\begin{aligned}r^T(x) &= r^0(x) = qr^N(x) \\ &= \beta R^N(x) - (\beta/\gamma)Q^{-1}\gamma R^0(x) \\ &= -(\beta/Q)R^T(x)\end{aligned}\tag{90}$$

so that  $\delta = -\beta/Q$ , as claimed. We must consider two subcases, according as we are still in the continuation loop in program 11 or are in the completion loop.

### Subcase 2a: still in the continuation loop

We have

$$r^0(x)(\text{new}) = r^T(x) = -(\beta/Q)R^T(x) = \delta R^N(x)(\text{new})$$

and

$$r^N(x)(\text{new}) = xr^N(x)(\text{old}) = \gamma x R^0(x)(\text{old}) = \gamma R^0(x)(\text{new}),$$

satisfying the conditions assumed for conclusion of a continuation loop.

Subcase 2b: in the completion loop

We have

$$\begin{aligned} r^0(x)(\text{new}) &= xr^N(x)(\text{old}) = \gamma x R^0(x)(\text{old}) = \gamma R^0(x)(\text{new}) \\ r^N(x)(\text{new}) &= r^T(x) = -(a/Q)R^T(x) = \delta R^N(x) \end{aligned}$$

satisfying the conditions assumed for conclusion of a completion loop.

Since initially the programs begin with

$$r^0(x) = R^0(x)$$

and

$$r^N(x) = R^N(x)$$

the assumptions for both case 1 and case 2 are always met. Thus, at every iteration  $j$  the polynomials  $a^j(x)$ ,  $b^j(x)$ , and  $r^j(x)$  produced by program 11 differ from those produced by program 12 only by a scale factor (though of course the roles of  $r^0(x)$  and  $r^N(x)$ , etc., are sometimes reversed).

We have now shown that program 5 (Mills' algorithm) is equivalent to program 11, which is equivalent to program 12, which is equivalent to program 8, which in turn is equivalent to program 7, which is an expanded version of program 6 (Berlekamp's algorithm).

We conclude that Mills' algorithm and the Berlekamp-Massey algorithm may be viewed as variants of Euclid's algorithm which are equivalent in the sense that partial results produced by one algorithm can be mapped directly into partial results produced by the other.

To complete this section we display table 2 showing the polynomials  $r^T(x)$  and  $R^T(x)$  defined at each iteration  $j$  for examples 16 and 17 using programs 11 and 12, respectively. Also shown are the values of the polynomials  $r^0(x)$ ,  $r^N(x)$ ,  $R^0(x)$ , and  $R^N(x)$  at the beginning of the iteration (i.e., these are the values determined for these polynomials during the  $j$ -1<sup>st</sup> iteration). From the table it is readily observed that the relations (89) and (90) hold for  $r^T(x)$  and  $R^T(x)$  determined at iterations following execution of a completion loop and following execution of the continuation loop, respectively.

Table 2. A Comparison of Outputs from Programs 9 ( $r^T(x)$ )  
and 10 ( $R^T(x)$ )

Program 11			Program 12		
j	$\begin{cases} r^0(x) \\ r^N(x) \\ r^T(x) \end{cases}$	$\begin{cases} \beta \\ \gamma \\ \delta \end{cases}$		$\begin{cases} R^0(x) \\ R^N(x) \\ R^T(x) \end{cases}$	
1	$x$ $10x^6+7x^5+9x^4+8x^3+2x^2+9x$ $5x^6+9x^5+10x^4+4x^3+x^2$	1 1 1	$x$ $10x^6+7x^5+9x^4+8x^3+2x^2+9x$ $5x^6+9x^5+10x^4+4x^3+x^2$		
2	$5x^6+9x^5+10x^4+4x^3+x^2$ $10x^7+7x^6+9x^5+8x^4+2x^3+9x^2$ $5x^7+3x^6+8x^5+3x^4+5x^3$	1 1 6	$10x^7+7x^6+9x^5+8x^4+2x^3+9x^2$ $5x^6+9x^5+10x^4+4x^3+x^2$ $10x^7+6x^6+5x^5+6x^4+10x^3$		
3	$10x^8+7x^7+9x^6+8x^5+2x^4+9x^3$ $5x^7+3x^6+8x^5+3x^4+5x^3$ $10x^8+9x^7+8x^6+9x^5+x^4$	1 6 1	$10x^8+7x^7+9x^6+8x^5+2x^4+9x^3$ $10x^7+6x^6+5x^5+6x^4+10x^3$ $10x^8+9x^7+8x^6+9x^5+x^4$		
4	$10x^8+9x^7+8x^6+9x^5+x^4$ $5x^8+3x^7+8x^6+3x^5+5x^4$ $9x^8+4x^7+2x^6+4x^5$	1 6 1	$10x^8+6x^7+5x^6+6x^5+10x^4$ $10x^8+9x^7+8x^6+9x^5+x^4$ $9x^8+4x^7+2x^6+4x^5$		
5	$5x^9+3x^8+8x^7+3x^6+5x^5$ $9x^8+4x^7+2x^6+4x^5$ $5x^9+0x^8+3x^7+6x^6$	6 1 6	$10x^9+6x^8+5x^7+6x^6+10x^5$ $9x^8+4x^7+2x^6+4x^5$ $10x^9+0x^8+6x^7+x^6$		
6	$5x^9+0x^8+3x^7+6x^6$ $9x^9+4x^8+2x^7+4x^6$ $8x^9+5x^8$	6 1 4	$9x^9+4x^8+2x^7+4x^6$ $10x^9+0x^8+6x^7+x^6$ $2x^9+4x^8$		

## 7.5 COMPARISONS

In this section some comparisons are made among the various algorithms which have been treated so far. We first compare, briefly, the versions of the Berlekamp-Massey algorithm that have been discussed; second, we make comparisons among the different versions of the Euclidean algorithm; finally, we make comparisons between the two classes and consider whether there is any choice to be made between programs 8 and 12.

The three versions of the Berlekamp-Massey algorithm which have been discussed are represented by programs 6, 7, and 8. All three programs solve the key equation (14) for  $\Lambda(x)$ ; programs 7 and 8 also provide  $\Omega(x)$  at the cost of more multiplications and storage for  $a(x)$ . Program 6 requires computation of the discrepancy  $d$  at every iteration by a vector inner product calculation whose length grows at each iteration. This would be highly undesirable if the algorithm were to be implemented in a VLSI systolic array. Programs 7 and 8 avoid this calculation by retaining, instead, an additional trio of polynomials  $r^N(x)$ ,  $r^O(x)$ , and  $r^T(x)$ .

Program 8 is more efficient than program 7 in that the updates of the old polynomials  $r^O(x)$ ,  $a^O(x)$ ,  $b^O(x)$  in lines 15-17 do not require a multiplication. However, both programs may be unsuitable for VLSI implementation. Program 7 usually requires computation of a finite field inverse  $d^{-1}$  at alternate iterations, while program 8 requires a finite field division  $r_j^N/r_j^O$  at every iteration. Both operations are considered difficult to implement in VLSI. In section 8.1 we examine Burton's enhancement of the Berlekamp-Massey algorithm. This modification obviates the need for computing finite field inverses or performing finite field division



within the Berlekamp-Massey algorithm. (Of course, a division is still required outside the algorithm if Forney's formula (16) is used to calculate the error magnitudes.)

The Euclidean decoding algorithms under consideration are represented by programs 4, 5, 11, and 12. It is clear that programs analogous to 11 and 12 can be constructed for the Japanese algorithm of program 4. Both programs 4 and 5 suffer certain deficiencies compared to programs 11 and 12 and the Berlekamp-Massey programs: they require polynomial division, itself an iterative algorithm; in certain situations they can have problems with termination; and there is a constant irksome need to determine the degrees of polynomials and vary action accordingly.

The quotient polynomials  $q(x)$  in programs 4 and 5 are usually, though not always, linear. On the average, one polynomial division of the Euclidean algorithm is equated with two iterations of the Berlekamp-Massey algorithm. But when we break the polynomial division of Mills' algorithm down into its component partial divisions in program 11, the number of iterations becomes  $2t$  for both algorithms. The difference is that each pair of iterations in the Berlekamp-Massey algorithm consists of two nearly identical steps, whereas each pair of partial divisions in the Japanese or Mills' algorithms consists of two distinct steps, clearly favoring the former.

Termination in programs 4 and 5 is correctly determined if the number of errors does not exceed  $t$ , the underlying assumption. However, in the Berlekamp-Massey algorithm, if more than  $t$  errors have occurred, the length  $\ell$  of the shift-register will sometimes,

though not often, exceed  $t$ , indicating that uncorrectable errors have occurred. Clearly, this is useful information which may be lost in programs 4 and 5. Program 12, however, does and program 11 may retain this information. In program 11, the degree of  $\Lambda(x)$  may have to be tested, for  $^\circ$  in this program is not a shift-register length.

All of these programs can also be used with arbitrary (nonsynchrone) sequences outside the decoding context. However, for programs 4 and 5, there is no certain way, with an arbitrary input sequence, of knowing when to halt the algorithm. (The other algorithms are terminated correctly by defining  $2t$  to be the sequence length.) Consider the following example.

Example 18: Let  $GF(19)$  be generated by the primitive root 2. Find shortest length LFSR's to generate the sequences

$$s_1 : 14, 7, 12, 15, 7, 15, 12, 7, 14, 6$$

and

$$s_2 : 6, 14, 7, 12, 15, 7, 15, 12, 7, 14.$$

The Berlekamp-Massey algorithm, with  $2t = 10$ , finds the solution

$$\Lambda(x) = 2x^6 + 4x^4 + 6x^3 + 15x^2 + 9x + 1$$

for sequence  $s_1$ . Programs 11 and 12, with  $2t = 10$ , find scalar multiples of this same solution:

$$14\Lambda(x) = 9x^6 + 18x^4 + 8x^3 + x^2 + 12x + 14.$$

and

$$3\Lambda(x) = 6x^6 + 12x^4 + 18x^3 + 7x^2 + 8x + 3.$$

However, programs 4 and 5, with  $t = 5$ , terminate too soon with the polynomial

$$b(x) = 15x^4 + 2x^3 + 16x^2 + 2x + 15.$$

If allowed to continue for one more iteration (e.g., by setting  $t = 6$ ) both programs find a correct (though different) solution.

When the reversal input sequence  $s_2$  is used, both programs 4 and 5 terminate correctly if  $t$  is chosen to be 5, but produce an incorrect result if  $t$  is set equal to 6. Thus, there is no safe way to use these programs with an arbitrary input sequence. (The programs terminate correctly if the sequence is repeated once and  $t$  is taken to be its original length 10.) Programs 8 and 12 have no difficulty with sequence  $s_2$ .

The third objection to programs 4 and 5 is the constant need for determining the degrees of the polynomials used in the algorithms, and for varying the action taken accordingly. Such a determination and comparison is implicit in each execution of (84).

When used to find a minimum length LFSR which generates an arbitrary sequence, even program 11 can have a problem, at termination, in determining the correct degree of the shift-register polynomial. (Recall that  $\ell$  in program 11 does not denote the shift-register length.) A correct solution for the sequence  $s_2$  of example 18 is

$$\Lambda(x) = 0x^5 + x^4 + 9x^3 + 15x^2 + 9x + 1.$$

The proper shift-register length is 5, not 4, that is, the last stage must be included even though it is not tapped. Program 11 finds a correct shift-register connection polynomial (a scalar multiple of  $\Lambda(x)$ ), but is unable to tell the correct shift-register length.

We conclude that for three substantial reasons, programs 4 and 5 are not competitive with programs 7, 8, and 12. Program 11 also seems to be out of contention; there is no reason to execute two distinct equally complex loops instead of executing one of them twice. Thus, we are left with comparing programs 8 and 12.

Between these two programs there would seem to be no preference. Both have the flaw that a finite field division is required. This we remove in section 8 by applying Burton's innovation. Program 8 produces a monic error locator polynomial  $\Lambda(x)$ . This may possibly give some advantage to program 8, depending on the method chosen to complete the error correction. However, as we have seen, neither the Chien search nor Forney's formula (16)

benefits from the use of a monic error locator polynomial. Furthermore, incorporation of Burton's enhancement eliminates the monicity advantage, if there is one.

Table 3 summarizes the estimates of the number of multiplications required for each of the programs. All basically require on the order of  $4t^2$  multiplications to find  $\Delta(x)$  when  $t$  errors have occurred. Program 6 can get by with  $2t^2$  multiplications if  $b^0(x)$  is not normalized, but does not provide  $Q(x)$ , and involves an unacceptable delay in the computation of the discrepancies. Program 7 becomes program 8 when the normalization of  $b^0(x)$ ,  $a^0(x)$ , and  $r^0(x)$  is omitted. All programs except program 6 require  $2t$  basic time units for correction of  $t$  errors, where a basic time unit includes the time required for a finite field division (or inversion), a multiplication, and a subtraction.

Table 3. Number of Multiplications Required for Obtaining  $\Delta(x)$  in the Presence of  $t$  Errors

Program	Number of Multiplications	Remarks
4	$4t^2$	
5	$5t^2$	1
6	$2.5t^2$	2
7	$6t^2$	
8	$4t^2$	
11	$4t^2$	
12	$4t^2$	

1 can be reduced to  $4t^2$  by dropping terms from  $r(x)$

2 can be reduced to  $2t^2$  by omitting normalization of  $b^0(x)$

## SECTION 8

### INVERSIONLESS DECODING

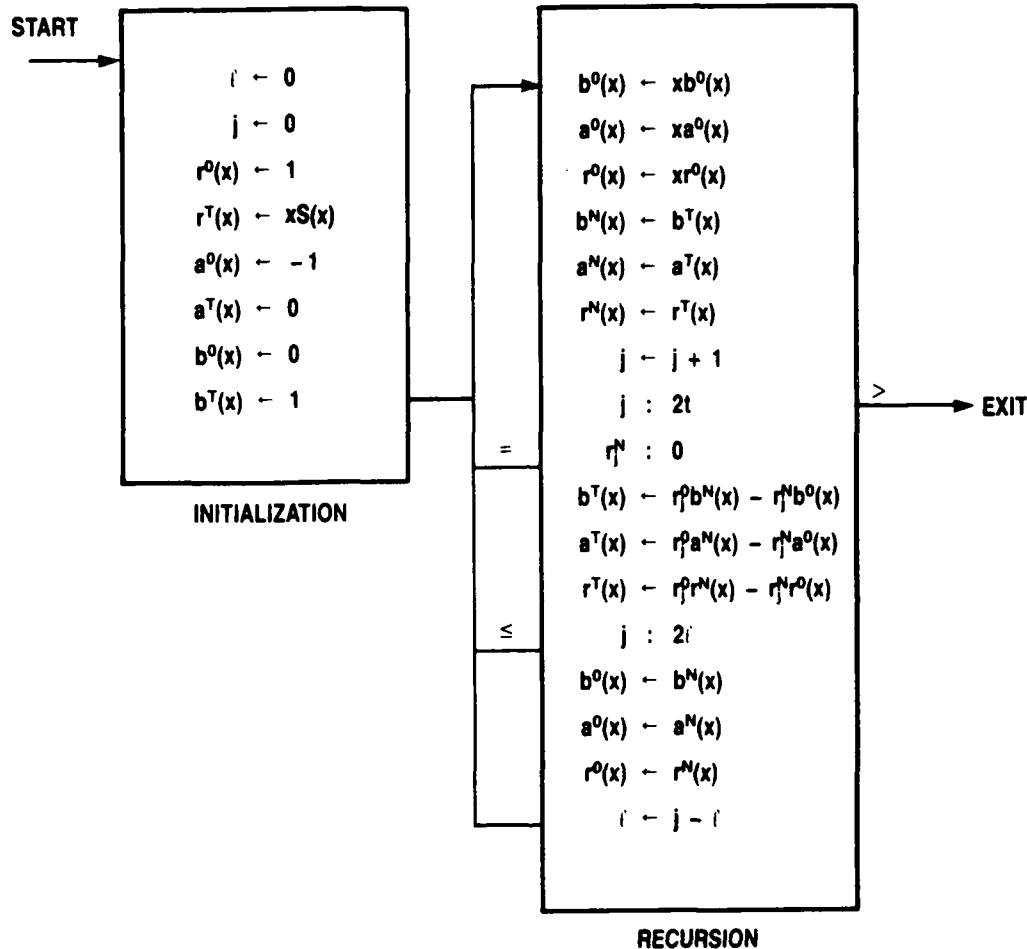
All programs which have been considered so far require inversion of finite field elements or, equivalently, division of finite field elements. These operations are generally considered to be difficult to implement efficiently in VLSI. In this section we consider algorithm modifications which avoid explicit computation of inverses and division at the cost of further multiplications. In section 8.1 we examine Burton's [27] inversionless variant of the Berlekamp-Massey algorithm. In section 8.2 we look at analogous methods for avoiding the computation of inverses in the Euclidean decoding algorithms.

#### 8.1 BURTON'S ALGORITHM

Burton [27] has given a modification to Berlekamp's algorithm which eliminates the usual inversion of the discrepancy  $d$  at each iteration, or, equivalently, the division  $r_j^N/r_j^0$  at each iteration in Citron's version (program 8).

At line 11 of the recursion in program 8 we want to compute the new shift register polynomial by

$$b^T(x) = b^N(x) - (r_j^N/r_j^0)b^0(x) \quad (91)$$



INPUT: SYNDROME POLYNOMIAL  $S(x)$ , INTEGER  $t$

OUTPUT:  $\gamma^{\wedge}(x) = b^N(x)$ ,  $\gamma^{\Omega}(x) = a^N(x)/x$

Program 13. BURTON'S ALGORITHM

without having to divide. Multiplying (91) by  $r_j^0$  gives

$$r_j^0 b^T(x) = r_j^0 b^N(x) - r_j^N b^0(x).$$

Burton accepts  $r_j^0 b^T(x)$  as the new shift register polynomial in place of  $b^T(x)$  as defined by (91), and similarly for  $a^T(x)$  and  $r^T(x)$ , giving program 13. At the termination of program 13 we end up with  $a^{2t}(x)/x = \gamma \Omega(x)$  and  $b^{2t}(x) = \gamma \Lambda(x)$  for some nonzero field element  $\gamma$ . This is acceptable, since neither the Chien search for the error locations nor Forney's calculation of the error magnitudes is affected. Citron [22] has also used Burton's modification to obtain an inversionless algorithm equivalent to program 13.

Example 19: Reed-Solomon 3-error-correcting code over  $GF(11)$  with  $\alpha = 2$ .

$t = 3$ ; Let  $c(x) = 0$ ,  $v(x) = e(x) = 6x^9 + 5x^8 + 3x^3$ .  
 $S(x) = 10x^5 + 7x^4 + 9x^3 + 8x^2 + 2x + 9$

$j = 0$ ,  $\ell = 0$

$$\begin{aligned} r^N(x) &= xS(x) = 10x^6 + 7x^5 + 9x^4 + 8x^3 + 2x^2 + 9x \\ r^0(x) &= x \\ b^N(x) &= 1 \\ b^0(x) &= 0 \\ a^N(x) &= 0 \\ a^0(x) &= -x = 10x \end{aligned}$$



$$j = 1, \ell = 0: r_j^N = 9, r_j^0 = 1$$

$$r^N(x) \leftarrow r^N(x) - 9r^0(x) = 10x^6 + 7x^5 + 9x^4 + 8x^3 + 2x^2$$

$$r^0(x) \leftarrow xr^N(x) = 10x^7 + 7x^6 + 9x^5 + 8x^4 + 2x^3 + 9x^2$$

$$b^N(x) \leftarrow b^N(x) - 9b^0(x) = 1$$

$$b^0(x) \leftarrow xb^N(x) = x$$

$$a^N(x) \leftarrow a^N(x) - 9a^0(x) = 9x$$

$$a^0(x) \leftarrow xa^N(x) = 0$$

$$j = 2, \ell = 1: r_j^N = 2, r_j^0 = 9$$

$$r^N(x) \leftarrow 9r^N(x) - 2r^0(x) = 2x^7 + 10x^6 + x^5 + 10x^4 + 2x^3$$

$$r^0(x) \leftarrow xr^0(x) = 10x^8 + 7x^7 + 9x^6 + 8x^5 + 2x^4 + 9x^3$$

$$b^N(x) \leftarrow 9b^N(x) - 2b^0(x) = 9x + 9$$

$$b^0(x) \leftarrow xb^0(x) = x^2$$

$$a^N(x) \leftarrow 9a^N(x) - 2a^0(x) = 4x$$

$$a^0(x) \leftarrow xa^0(x) = 0$$

$$j = 3, \ell = 1: r_j^N = 2, r_j^0 = 9$$

$$r^N(x) \leftarrow 9r^N(x) - 2r^0(x) = 2x^8 + 4x^7 + 6x^6 + 4x^5 + 9x^4$$

$$r^0(x) \leftarrow xr^N(x) = 2x^9 + 10x^8 + x^7 + 10x^6 + 2x^5$$

$$b^N(x) \leftarrow 9b^N(x) - 2b^0(x) = 9x^2 + 4x + 4$$

$$b^0(x) \leftarrow xb^N(x) = 9x^3 + 9x^2$$

$$a^N(x) \leftarrow 9a^N(x) - 2a^0(x) = 3x$$

$$a^0(x) \leftarrow xa^N(x) = 4x^2$$

$$j = 4, \ell = 2: r_j^N = 9, r_j^0 = 2$$

$$r^N(x) \leftarrow 2r^N(x) - 9r^0(x) = 8x^8 + 6x^7 + 3x^6 + 6x^5$$

$$r^0(x) \leftarrow xr^N(x) = 2x^9 + 10x^8 + x^7 + 10x^6 + 2x^5$$

$$b^N(x) \leftarrow 2b^N(x) - 9b^0(x) = 3x^2 + 4x + 8$$

$$b^0(x) \leftarrow xb^N(x) = 9x^3 + 9x^2$$

$$a^N(x) \leftarrow 2a^N(x) - 9a^0(x) = 8x^2 + 6x$$

$$a^0(x) \leftarrow xa^N(x) = 4x^3$$

$$j = 5, \ell = 2: r_j^N = 6, r_j^0 = 2$$

$$r^N(x) \leftarrow 2r^N(x) - 6r^0(x) = 10x^9 + 0x^8 + 6x^7 + x^6$$

$$r^0(x) \leftarrow xr^N(x) = 8x^9 + 6x^8 + 3x^7 + 6x^6$$

$$b^N(x) \leftarrow 2b^N(x) - 6b^0(x) = x^3 + 7x^2 + 8x + 5$$

$$b^0(x) \leftarrow xb^N(x) = 3x^3 + 4x^2 + 8x$$

$$a^N(x) \leftarrow 2a^N(x) - 6a^0(x) = 9x^3 + 5x^2 + x$$

$$a^0(x) \leftarrow xa^N(x) = 8x^3 + 6x^2$$

$$j = 6, \ell = 3: r_j^N = 1, r_j^0 = 6$$

$$r^N(x) \leftarrow 6r^N(x) - r^0(x) = 8x^9 + 5x^8 + 0x^7$$

$$r^0(x) \leftarrow xr^N(x) = 8x^{10} + 6x^9 + 3x^8 + 6x^7$$

$$b^N(x) \leftarrow 6b^N(x) - b^0(x) = 3x^3 + 5x^2 + 7x + 8$$

$$b^0(x) \leftarrow xb^N(x) = 3x^4 + 4x^3 + 8x^2$$

$$a^N(x) \leftarrow 6a^N(x) - a^0(x) = 2x^3 + 2x^2 + 6x$$

$$a^0(x) \leftarrow xa^N(x) = 8x^4 + 6x^3$$

$j = 7$ , stop.  $\gamma = 8$ ,  $\gamma^{-1} = 7$

$$\begin{aligned} \Lambda(x) &= \gamma^{-1}b^N(x) = 10x^3 + 2x^2 + 5x + 1 \\ \Omega(x) &= \gamma^{-1}a^N(x)/x = 3x^2 + 3x + 9 \\ x^2tA(x) &= \gamma^{-1}r^N(x)/x = x^8 + 2x^7 \\ \lambda(x) &= x^2 + 2x \end{aligned}$$

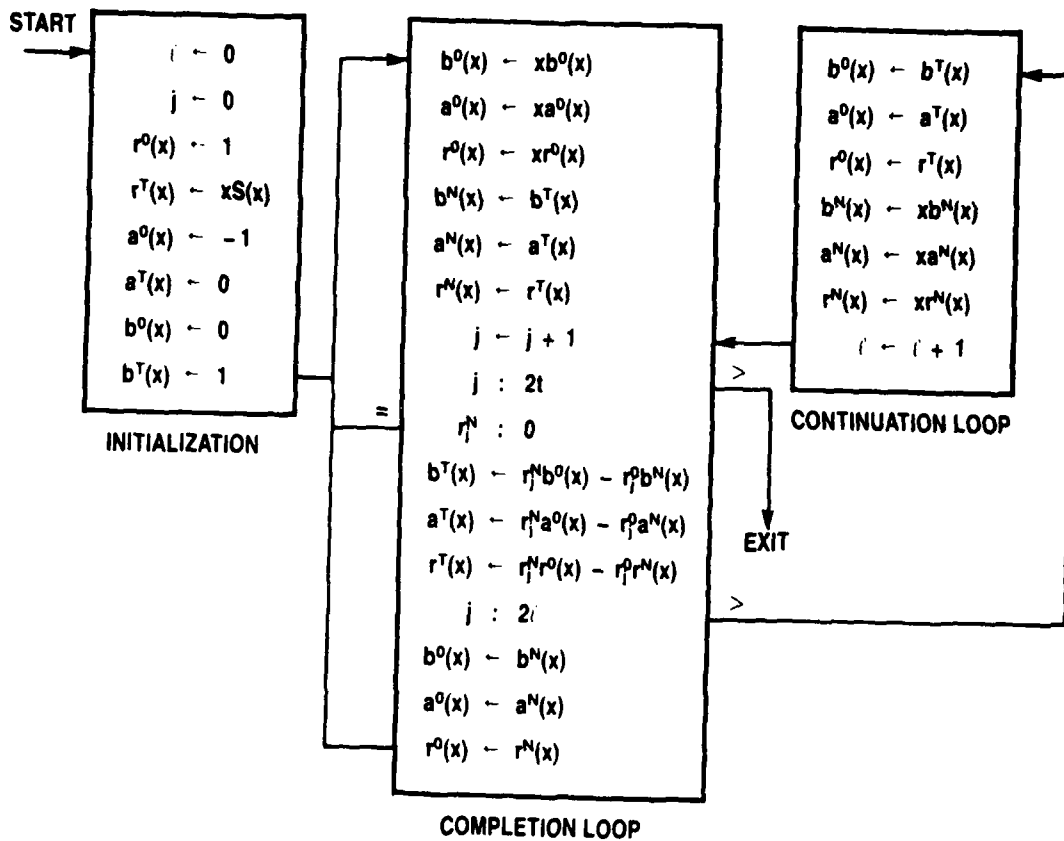
Burton's modification almost doubles the number of multiplications. We require  $4t^2 + 2t$  multiplications to update  $r^T(x)$ ,  $2t^2 + 2t$  for  $b^T(x)$ , and  $2t^2 - 2t + 2$  for  $a^T(x)$  for a total on the order of  $8t^2$  multiplications.

## 8.2 INVERSIONLESS EUCLIDEAN ALGORITHMS

The Japanese decoding algorithm and Mills' algorithm can be put into inversionless form by Burton's technique if we first break the polynomial divisions down explicitly into their partial divisions as was done in program 11. To eliminate the finite field division in the completion loop of program 11, we delete statement 10 (which defined  $q$ ) and replace statement 11 by

$$b^T(x) \leftarrow r_j^N b^0(x) - r_j^0 b^N(x) \quad (92)$$

etc., resulting in program 14. Sugiyama, et al. [12] have used Burton's technique to yield an inversionless variant of their algorithm which is equivalent to program 14. Program 14 requires on the order of  $8t^2$  multiplications. Shao, et al. [28] have also presented an inversionless variant of the Japanese algorithm which is similar to program 14.



INPUT: SYNDROME POLYNOMIAL  $S(x)$ , INTEGER  $t$   
 OUTPUT:  $\gamma \lambda(x) = b^N(x)$ ,  $\gamma \Omega(x) = a^N(x)/x$

Program 14. INVERSIONLESS MILLS' ALGORITHM

Finally, let us apply Burton's technique to the simplified Mill's decoder of program 12. Analogous to the changes made in program 8 to obtain program 13, we delete statement 10 of the recursion which defined  $q$ , and modify statements 11-13, replacing statement 11 by (92), etc., as in program 14. These changes produce program 15. It is manifest that programs 13 and 15 are equivalent, the only difference being the signs of the r.h.s. of statements 10-12 of the recursion. Example 20 shows how this change affects the partial results.

Example 20: Reed-Solomon 3-error-correcting code over  $GF(11)$  with  $\alpha = 2$ .

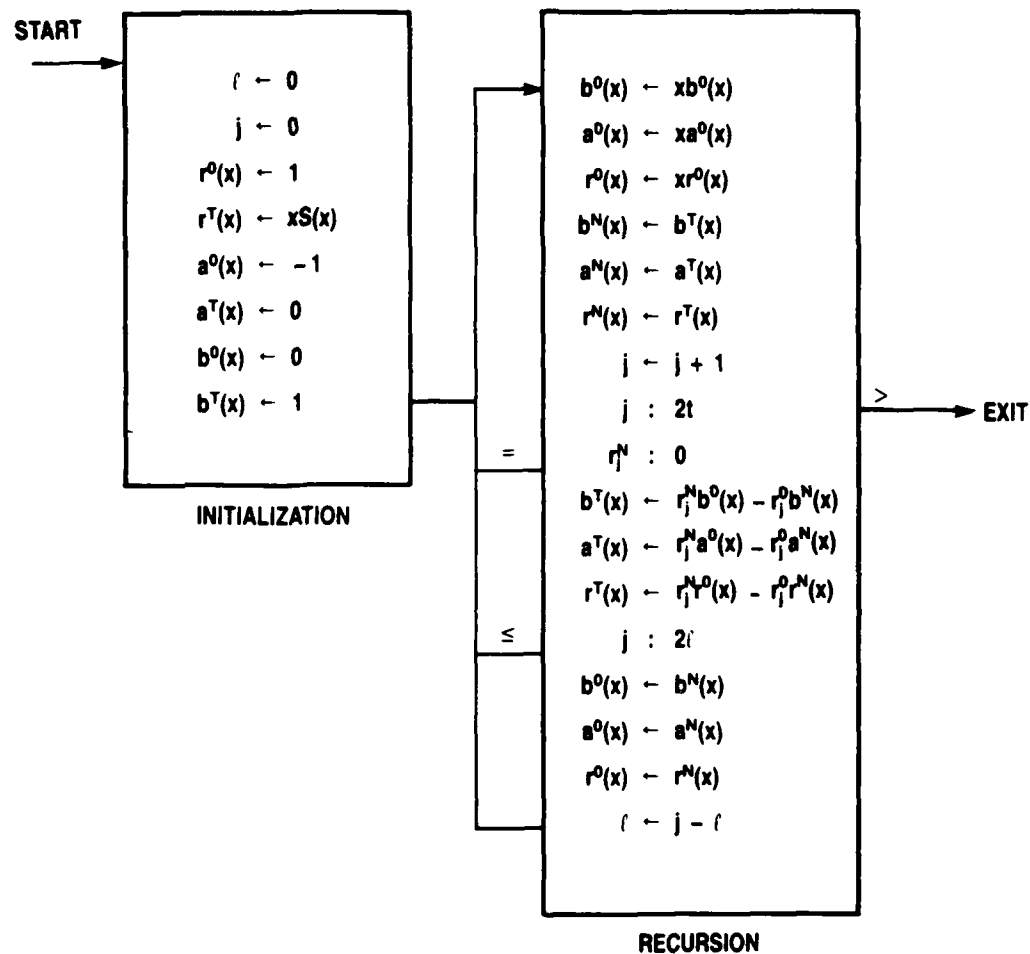
$t = 3$ ; Let  $c(x) = 0$ ,  $v(x) = e(x) = 6x^9 + 5x^8 + 3x^3$ .  
 $S(x) = 10x^5 + 7x^4 + 9x^3 + 8x^2 + 2x + 9$ .

$j = 0, \ell = 0$

$$\begin{aligned} r^N(x) &= xS(x) = 10x^6 + 7x^5 + 9x^4 + 8x^3 + 2x^2 + 9x \\ r^0(x) &= x \\ b^N(x) &= 1 \\ b^0(x) &= 0 \\ a^N(x) &= 0 \\ a^0(x) &= -x = 10x \end{aligned}$$

$j = 1, \ell = 0: r_j^N = 9, r_j^0 = 1$

$$\begin{aligned} r^N(x) + 9r^0(x) - r^N(x) &= x^6 + 4x^5 + 2x^4 + 3x^3 + 9x^2 \\ r^0(x) + xr^N(x) &= 10x^7 + 7x^6 + 9x^5 + 8x^4 + 2x^3 + 9x^2 \\ b^N(x) + 9b^0(x) - b^N(x) &= 10 \\ b^0(x) + xb^N(x) &= x \\ a^N(x) + 9a^0(x) - a^N(x) &= 2x \\ a^0(x) + xa^N(x) &= 0 \end{aligned}$$



INPUT: SYNDROME POLYNOMIAL  $S(x)$ , INTEGER  $t$

OUTPUT:  $\gamma \wedge(x) = b^N(x)$ ,  $\gamma \Omega(x) = a^N(x)/x$

Figure 15. BURTONIZED MILLS' ALGORITHM — FINAL VERSION

$$j = 2, \ell = 1: r_j^N = 9, r_j^0 = 9$$

$$r^N(x) + 9r^0(x) - 9r^N(x) = 2x^7 + 10x^6 + x^5 + 10x^4 + 2x^3$$

$$r^0(x) + xr^0(x) = 10x^8 + 7x^7 + 9x^6 + 8x^5 + 2x^4 + 9x^3$$

$$b^N(x) + 9b^0(x) - 9b^N(x) = 9x + 9$$

$$b^0(x) + xb^0(x) = x^2$$

$$a^N(x) + 9a^0(x) - 9a^N(x) = 4x$$

$$a^0(x) + xa^0(x) = 0$$

$$j = 3, \ell = 1: r_j^N = 2, r_j^0 = 9$$

$$r^N(x) + 2r^0(x) - 9r^N(x) = 9x^8 + 7x^7 + 5x^6 + 7x^5 + 2x^4$$

$$r^0(x) + xr^N(x) = 2x^8 + 10x^7 + x^6 + 10x^5 + 2x^4$$

$$b^N(x) + 2b^0(x) - 9b^N(x) = 2x^2 + 7x + 7$$

$$b^0(x) + xb^N(x) = 9x^2 + 9x$$

$$a^N(x) + 2a^0(x) - 9a^N(x) = 8x$$

$$a^0(x) + xa^N(x) = 4x^2$$

$$j = 4, \ell = 2: r_j^N = 2, r_j^0 = 2$$

$$r^N(x) + 2r^0(x) - 2r^N(x) = 8x^8 + 6x^7 + 3x^6 + 6x^5$$

$$r^0(x) + xr^0(x) = 2x^9 + 10x^8 + x^7 + 10x^6 + 2x^5$$

$$b^N(x) + 2b^0(x) - 2b^N(x) = 3x^2 + 4x + 8$$

$$b^0(x) + xb^0(x) = 9x^3 + 9x^2$$

$$a^N(x) + 2a^0(x) - 2a^N(x) = 8x^2 + 6x$$

$$a^0(x) + xa^0(x) = 4x^3$$

$$j = 5, \nu = 2: r_j^N = 6, r_j^0 = 2$$

$$r^N(x) + 6r^0(x) - 2r^N(x) = x^9 + 0x^8 + 5x^7 + 10x^6$$

$$r^0(x) + xr^N(x) = 8x^9 + 6x^8 + 3x^7 + 6x^6$$

$$b^N(x) + 6b^0(x) - 2b^N(x) = 10x^3 + 4x^2 + 3x + 6$$

$$b^0(x) + xb^N(x) = 3x^3 + 4x^2 + 8x$$

$$a^N(x) + 6a^0(x) - 2a^N(x) = 2x^3 + 6x^2 + 10x$$

$$a^0(x) + xa^N(x) = 8x^3 + 6x^2$$

$$j = 6, \nu = 3: r_j^N = 10, r_j^0 = 6$$

$$r^N(x) + 10r^0(x) - 6r^N(x) = 8x^9 + 5x^8 + 0x^7$$

$$r^0(x) + xr^N(x) = 8x^{10} + 6x^9 + 3x^8 + 6x^7$$

$$b^N(x) + 10b^0(x) - 6b^N(x) = 3x^3 + 5x^2 + 7x + 8$$

$$b^0(x) + xb^N(x) = 3x^4 + 4x^3 + 8x^2$$

$$a^N(x) + 10a^0(x) - 6a^N(x) = 2x^3 + 2x^2 + 6x$$

$$a^0(x) + xa^N(x) = 8x^4 + 6x^3$$

$$j = 7, \text{ stop. } \nu = 8, \nu^{-1} = 7$$

$$\lambda(x) = \nu^{-1}b^N(x) = 10x^3 + 2x^2 + 5x + 1$$

$$\rho(x) = \nu^{-1}a^N(x)/x = 3x^2 + 3x + 9$$

$$x^2t_A(x) = \nu^{-1}r^N(x)/x = x^8 + 2x^7$$

$$f(x) = x^2 + 2x$$



## SECTION 9

### DECODING ERASURES

In this section earlier results are extended to include the decoding of erasures in addition to errors, where an erasure is an error whose location is already known to the decoder. A BCH  $t$ -error correcting code, with minimum distance  $2t + 1$ , is capable of correcting any combination of  $v$  errors and  $\mu$  erasures for which  $2v + \mu \leq 2t$ . Forney [9] first showed that by employing modified syndromes one can still solve for the error locator polynomial in the presence of erasures. Blahut [29] showed that the errata locator polynomial (where an erratum is either an error or an erasure) can be calculated directly (without first finding the error locator polynomial) by initializing Berlekamp's algorithm with the erasure locator polynomial. In this section we combine these results to give a program which provides both the errata locator polynomial and the errata evaluator polynomial. Errata magnitudes can then be calculated by Forney's formula (16) or by the new formula (24).

As in section 3, we assume a BCH code designed to correct  $t$  errors in a codeword of length  $n = q^m - 1$  for  $q$  a power of a prime. Let  $c(x)$  represent the transmitted codeword polynomial and  $e(x)$  be an error polynomial. In addition, let  $d(x)$  represent the channel erasure polynomial. The received codeword polynomial is now

$$v(x) = c(x) + d(x) + e(x).$$

We define  $2t$  error syndromes by

$$\begin{aligned} S_j &= v(\alpha^j) = c(\alpha^j) + d(\alpha^j) + e(\alpha^j) \\ &= d(\alpha^j) + e(\alpha^j) \quad (j = 1, \dots, 2t) \end{aligned}$$

where  $\alpha$  is a primitive element of  $GF(q^m)$ .

Suppose  $v$  errors and  $u$  erasures, where  $2v + u \leq 2t$ , have occurred during transmission. We define  $v$  unknown error locations  $X_\ell$ , where  $X_\ell$  is the field element of  $GF(q^m)$  associated with the  $\ell$ th error location, and  $v$  unknown error magnitudes  $Y_\ell$ , where  $Y_\ell \neq 0$  and  $Y_\ell \in GF(q)$ . In addition, we now define  $u$  known erasure locations  $W_k \in GF(q^m)$ , where  $W_k$  is the field element associated with the  $k$ th erasure location, and  $u$  unknown erasure magnitudes  $V_k \in GF(q)$ . The  $W_k$  are always assumed to be distinct from the  $X_\ell$ .  $V_k$  is the difference between the transmitted symbol at location  $W_k$  and the symbol assumed for the  $k$ th erasure at the receiving end. Unlike  $Y_\ell$ ,  $V_k$  may assume the value 0. The  $2t$  syndromes are now given by the  $2t$  BCH decoding equations

$$\begin{aligned} S_j &= e(\alpha^j) + d(\alpha^j) = \sum_{\ell=1}^v Y_\ell X_\ell^j + \sum_{k=1}^u V_k W_k^j \\ &= E_j + D_j, \quad (j = 1, \dots, 2t). \end{aligned} \tag{93}$$

The error-and-erasure decoding problem for BCH codes is to solve this set of  $2t$  (nonlinear simultaneous) equations for the  $v$  unknown error locations  $X_\rho$ , the  $v$  unknown error magnitudes  $Y_\rho$ , and the  $\mu$  unknown erasure magnitudes  $V_k$ , given the  $2t$  syndromes  $S_j$  and the  $\mu$  erasure locations  $W_k$ . Forney's solution is to derive from the set of  $2t$  equations (93) a reduced set of  $2t - \mu$  equations of the form (9) which can be solved for the error locator polynomial  $\Lambda(x)$ .

If we define  $\Lambda(x)$  by (10), and  $E_j$  by

$$E_j = \sum_{\rho=1}^v Y_\rho X_\rho^j, \quad (j = 1, \dots, 2t)$$

then by a process identical to that which obtained equation (11) from (10) in section 3, we arrive at

$$0 = E_{j+v} + \sum_{i=1}^v \Lambda_i E_{j+v-i}, \quad (j = 1, \dots, 2t). \quad (94)$$

This set of  $2t$  simultaneous linear equations could be solved to obtain  $\Lambda(x)$  if we knew the  $E_j$ . However, we do not know the  $E_j$ , but only the  $S_j = E_j + D_j$ , where

$$D_j = \sum_{k=1}^{\mu} V_k W_k^j.$$

We now define the erasure locator polynomial as the monic polynomial having zeros at the inverse erasure locations  $W_k^{-1}$  for  $k = 1, \dots, \mu$ :

$$\kappa(x) = \prod_{k=1}^{\mu} (1 - W_k x) = 1 + \sum_{i=1}^{\mu} \kappa_i x^i. \quad (95)$$

(If  $\mu = 0$ ,  $\kappa(x)$  is defined as the zero-degree polynomial 1.) Forney uses the erasure locator polynomial  $\kappa(x)$  to define a set of  $2t - \mu$  modified syndromes  $T_j$  ( $j = \mu + 1, \dots, 2t$ ) and to derive a reduced set of  $2t - \mu$  equations from (94) in the modified syndromes  $T_j$  which can be solved for  $\Lambda(x)$ .

Let the modified syndromes be defined by

$$T_j = \sum_{i=0}^{\mu} \kappa_i S_{j-i}, \quad (j = \mu + 1, \dots, 2t). \quad (96)$$

By extension, defining  $S_j = 0$  for  $j$  outside the range  $(1, 2t)$  allows (96) to be used for defining  $T_j$  for  $j$  outside the range  $(\mu + 1, 2t)$ .

Now, since  $W_k^{-1}$  is a zero of  $\kappa(x)$  for  $k = 1, \dots, \mu$ , we have

$$\sum_{i=0}^{\mu} \kappa_i W_k^{j-i} = 0,$$

and

$$\sum_{i=0}^{\mu} \kappa_i D_{j-i} = 0.$$

Therefore,

$$T_j = \sum_{i=0}^{\mu} \kappa_i S_{j-i} = \sum_{i=0}^{\mu} \kappa_i (E_{j-i} + D_{j-i}) = \sum_{i=0}^{\mu} \kappa_i E_{j-i}. \quad (97)$$

We now multiply (94) by  $\kappa_\ell$  and sum  $\mu + 1$  successive equations, using (97) to obtain the set of  $2t - \mu$  equations

$$\begin{aligned} 0 &= \sum_{\ell=0}^{\mu} \kappa_\ell (E_{j+v-\ell} + \sum_{i=1}^v \Lambda_i E_{j+v-i-\ell}) \\ &= T_{j+v} + \sum_{i=1}^v \Lambda_i T_{j+v-i}, \quad (j = \mu + 1, \dots, 2t). \end{aligned} \quad (98)$$

This set of  $2t - \mu$  equations in  $v$  unknowns  $\Lambda_i$ , where  $2v \leq 2t - \mu$ , is exactly analogous to the set (11), and can be solved for  $\Lambda(x)$  by the Peterson-Gorenstein-Zierler algorithm exactly as in section 3.

The modified syndrome polynomial is defined analogously to  $S(x)$  by

$$\begin{aligned} T(x) &= \sum_{j=1}^{2t} T_j x^{j-1} \\ &= \left| \kappa(x) S(x) \right|_x^{2t} \end{aligned} \quad (99)$$

and the errata locator polynomial  $\Pi(x)$  is defined as the product of the erasure locator polynomial and the error locator polynomial:

$$\Pi(x) = \kappa(x) \Lambda(x). \quad (100)$$

We shall re-use  $\Omega(x)$  to denote the errata evaluator polynomial, which is defined by the key equation for erasure-and-error decoding:

$$\begin{aligned}\Omega(x) &= \left| \Pi(x)S(x) \right|_{x^{2t}} \\ &= \left| \Lambda(x)T(x) \right|_{x^{2t}}.\end{aligned}\quad (101)$$

Any of the programs (see, e.g., Berlekamp [8], pp. 229-231 and Sugiyama, et al. [30]) supplied in sections 4 - 8 can be used to decode both erasures and errors, yielding  $\Lambda(x)$  and  $\Omega(x)$ , if we first replace  $S(x)$  by  $T(x)$ , as computed by (99). The error locations can be determined by applying a Chien search either to  $\Lambda(x)$  or to  $\Pi(x)$ .  $\Pi(x)$  can be obtained from  $\Lambda(x)$  and  $\kappa(x)$ . Forney's formula (16) now becomes

$$Y_j = - \frac{\Omega(X_j^{-1})}{\Pi'(X_j^{-1})} \quad (102)$$

where  $Y_j$  and  $X_j$  are now interpreted as errata magnitudes and locations, and  $j$  runs from 1 to  $v + \mu$ .

However, Blahut [29] has pointed out that it is unnecessary to obtain  $\Lambda(x)$ . If, in Berlekamp's algorithm, the shift-register connection polynomial  $b(x)$  is initialized by the erasure locator polynomial  $\kappa(x)$ , then at termination this polynomial will yield  $\Pi(x)$  in place of  $\Lambda(x)$ . (In program 6 we initialize  $\ell$  and  $j$  by the number of erasures  $\mu$  and  $y(x)$  by  $\kappa(x)$ ; the length test (line 8 of the recursion) is modified to " $j + \mu : 2\ell$ "; the length specification (line 10) is changed to " $\ell + j - \ell + \mu$ "; and the modified syndromes  $T_j$  are used in place of the syndromes  $S_j$  at line 5 of the recursion.)

If we use one of the Euclideanized versions of the Berlekamp-Massey algorithm, we directly obtain the errata evaluator polynomial  $Q(x)$  as well as  $\Pi(x)$ . We show this in program 16, which is program 13 (Burton's algorithm) modified for handling erasures, and with  $a(x)$  superimposed on  $r(x)$ . In this program the integer  $\mu$  represents the number of erasures. If  $\mu = 0$ , the program functions like program 13. There is no confusion in superimposing  $a^N(x)$  and  $r^N(x)$ , since at iteration  $j$ ,  $r_i^N = 0$  for  $i < j$ ,  $i < j$ , and by (66)  $a_i^N = 0$  for  $i \geq \mu$ . When  $\mu > 0$ ,  $r_i^N$  may be nonzero for  $i < \mu$  at iteration  $j > \mu$ , but  $a_j^N = 0$ , so there is no problem in determining  $r_j^N$ .

There is a problem with  $r^0(x)$ , however. Note that  $r^0(x)$  is now initialized by 0, the sum of the initializations for  $a^0(x)$  and  $r^0(x)$  in program 13, but we still need  $r^0(x) = 1$  for the initial update of  $r^T(x)$ . Therefore, following Burton [27], we retain an additional variable  $\delta$  to represent the old discrepancy value. This is initialized as 1 and updated as the current discrepancy  $d$  at every length change. The variable  $\delta$  is not needed if  $a(x)$  and  $r(x)$  are not superimposed.

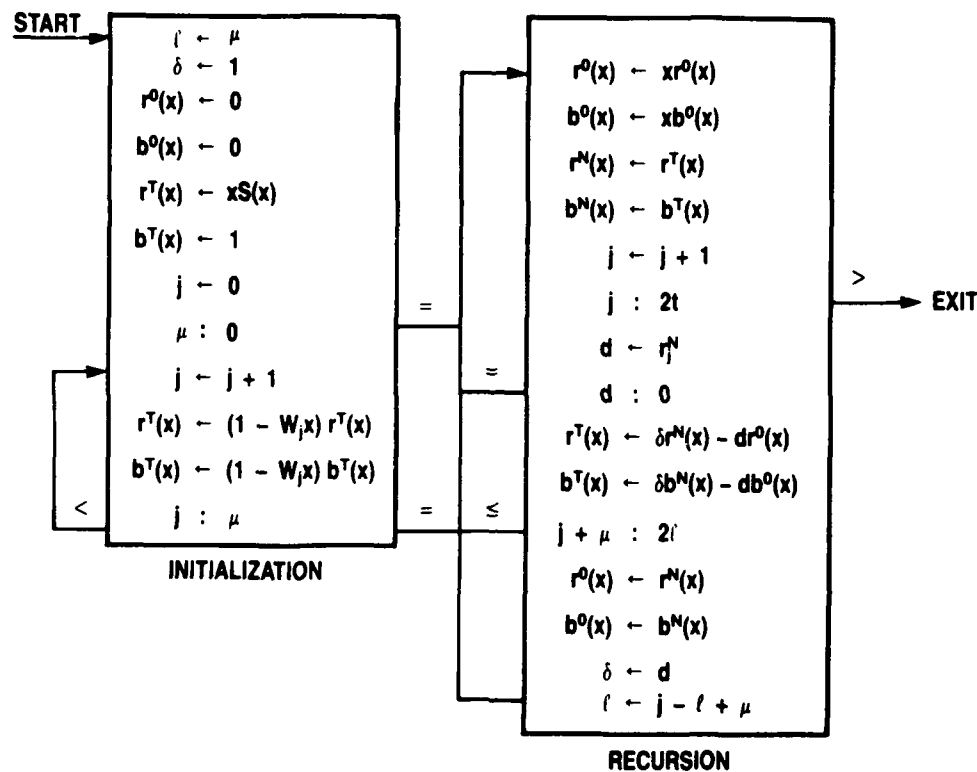
At termination we have

$$r^N(x) = a^N(x)(-1) + b^N(x)xS(x).$$

For  $b^N(x)$  to give  $\gamma\Pi(x)$ , we must have, by (101),

$$\gamma Q(x) = \left| (a^N(x) + r^N(x))/x \right|_{x^{2t}}$$

thus providing the motivation for superimposing the two polynomials.



INPUT: SYNDROME POLYNOMIAL  $S(x)$ , ERASURE LOCATIONS  $W_i$  (FIELD ELEMENTS)  
 NUMBER OF ERASURES  $\mu$ , INTEGER  $t$

OUTPUT:  $\gamma\Pi(x) = b^N(x)$ ,  $\gamma\Omega(x) = |r^N(x)/x|_{x^{2t}}$

Program 16. DECODING WITH ERASURES



If  $A(x)$  is defined analogously to (19) by

$$A(x) = \lfloor (\tau(x)S(x))/x^{2t} \rfloor \quad (103)$$

then at termination of program 16 we have  $A(x)$  given by  $\lfloor ((a^N(x) + r^N(x))/x)/x^{2t} \rfloor$ , which, since  $a(x)$  and  $r(x)$  are superimposed in program 16 is simply  $\lfloor r^N(x)/x^{2t+1} \rfloor$ . Error magnitudes can then be calculated by a revised version of (24):

$$Y_j = - \frac{\bar{A}(x_j)}{\bar{\Pi}'(x_j)} \cdot x_j^{n-d} \quad (104)$$

where  $n$  is the codeword length and  $d$  is the designed distance of the code.

For an example we again call upon the Reed-Solomon 3-error-correcting code over  $GF(11)$  with  $\alpha = 2$ .

Example 21: Reed-Soloman 3-error-correcting code over GF(11) with  
 $\alpha = 2$

$t = 3$ ; Let  $c(x) = 0$ ,  $e(x) = 3x^3 + 7$ ,  $d(x) = 6x^9 + 5x^8$ .

$$v(x) = c(x) + d(x) + e(x) = 6x^9 + 5x^8 + 3x^3 + 7$$

Since  $v(x)$  for this example is  $7 + v(x)$  for ex. 3, the  $S_j$  for this example are  $7 + S_j$  for example 3.

$$S(x) = 6x^5 + 3x^4 + 5x^3 + 4x^2 + 9x + 5$$

$$j = 0, \ell = \mu = 2, \delta = 1, r^0(x) = b^0(x) = 0$$

$$\begin{aligned} r^T(x) + xS(x) &= 6x^6 + 3x^5 + 5x^4 + 4x^3 + 9x^2 + 5x \\ b^T(x) &= 1 \end{aligned}$$

$$j = 1 : W_j = \alpha^8 = 3$$

$$\begin{aligned} r^T(x) + (1-3x)r^T(x) &= 4x^7 + 8x^6 + 10x^5 + 4x^4 + 10x^3 + 5x^2 + 5x \\ b^T(x) + (1-3x)b^T(x) &= 8x + 1 \end{aligned}$$

$$j = 2 : W_j = \alpha^9 = 6$$

$$\begin{aligned} r^T(x) + (1-6x)r^T(x) &= 9x^8 + 0x^7 + 3x^6 + 8x^5 + 10x^4 + 2x^3 + 8x^2 + 5x \\ b^T(x) + (1-6x)b^T(x) &= 7x^2 + 2x + 1 \end{aligned}$$

$$j = 3, \ell = 2 : r_j^N = 2, \delta = 1$$

$$\begin{aligned} r^N(x) + r^N(x) - 2r^0(x) &= 9x^8 + 0x^7 + 3x^6 + 8x^5 + 10x^4 + 2x^3 + 8x^2 + 5x \\ r^0(x) + xr^N(x) &= 9x^9 + 0x^8 + 3x^7 + 8x^6 + 10x^5 + 2x^4 + 8x^3 + 5x^2 \\ b^N(x) + b^N(x) - 2b^0(x) &= 7x^2 + 2x + 1 \\ b^0(x) + xb^N(x) &= 7x^3 + 2x^2 + x \end{aligned}$$

$$j = 4, \ell = 3 : r_j^N = 10, \delta = 2$$

$$\begin{aligned} r^N(x) &\leftarrow 2r^N(x) - 10r^0(x) = 9x^9 + 7x^8 + 3x^7 + 3x^6 + 4x^5 + 0x^4 + x^3 + 10x^2 + 10x \\ r^0(x) &\leftarrow xr^0(x) = 9x^{10} + 0x^9 + 3x^8 + 8x^7 + 10x^6 + 2x^5 + 8x^4 + 5x^3 \\ b^N(x) &\leftarrow 2b^N(x) - 10b^0(x) = 7x^3 + 5x^2 + 5x + 2 \\ b^0(x) &\leftarrow xb^0(x) = 7x^4 + 2x^3 + x^2 \end{aligned}$$

$$j = 5, \ell = 3 : r_j^N = 4, \delta = 2$$

$$\begin{aligned} r^N(x) &\leftarrow 8x^{10} + 7x^9 + 2x^8 + 7x^7 + 10x^6 + 0x^5 + x^4 + 4x^3 + 9x^2 + 9x \\ r^0(x) &\leftarrow 9x^{10} + 7x^9 + 3x^8 + 3x^7 + 4x^6 + 0x^5 + x^4 + 10x^3 + 10x^2 \\ b^N(x) &\leftarrow 5x^4 + 6x^3 + 6x^2 + 10x + 4 \\ b^0(x) &\leftarrow 7x^4 + 5x^3 + 5x^2 + 2x \end{aligned}$$

$$j = 6, \ell = 4 : r_j^N = 10, \delta = 4$$

$$\begin{aligned} r^N(x) &\leftarrow 8x^{10} + 2x^9 + 0x^8 + 9x^7 + 0x^6 + 0x^5 + 5x^4 + 4x^3 + 2x^2 + 3x \\ r^0(x) &\leftarrow 9x^{11} + 7x^{10} + 3x^9 + 3x^8 + 4x^7 + 0x^6 + x^5 + 10x^4 + 10x^3 \\ b^N(x) &\leftarrow 5x^4 + 7x^3 + 7x^2 + 9x + 5 \\ b^0(x) &\leftarrow 7x^5 + 5x^4 + 5x^3 + 2x^2 \end{aligned}$$

$$j = 7, \text{ stop: } \gamma = 5, \gamma^{-1} = 9$$

$$\pi(x) = \gamma^{-1}b^N(x) = x^4 + 8x^3 + 8x^2 + 4x + 1$$

$$\begin{aligned} Q(x) &= \gamma^{-1} \left| r^N(x)/x \right|_{x^{2t}} = \gamma^{-1}(5x^3 + 4x^2 + 2x + 3) \\ &= x^3 + 3x^2 + 7x + 5 \end{aligned}$$

$$x^{2t}A(x) = \gamma^{-1}(r^N(x)/x)$$

$$A(x) = 6x^3 + 7x^2 + 0x + 4$$

Chien Search:  $j$      $\alpha^j$      $\pi(\alpha^j)$

0	1	0
1	2	0
2	4	0
3	8	3
4	5	9
5	10	9
6	9	10
7	7	0
8	3	8
9	6	4

Inverse errata locations:  $\alpha^0, \alpha^7, \alpha^2, \alpha^1$

Errata locations:  $\alpha^0, \alpha^3, \alpha^8, \alpha^9$

#### Errata Magnitudes

a) evaluated by Forney's formula (16):

$$\pi^*(x) = 4x^3 + 2x^2 + 5x + 4$$

$$Y_1 = - \frac{Q(\alpha^0)}{\pi^*(\alpha^0)} = - \frac{1 + 3 + 7 + 5}{4 + 2 + 5 + 4} = - \frac{5}{4} = 7$$

$$Y_2 = - \frac{Q(\alpha^7)}{\pi^*(\alpha^7)} = - \frac{2 + 4 + 5 + 5}{8 + 10 + 2 + 4} = - \frac{5}{2} = 3$$

$$Y_3 = - \frac{Q(\alpha^2)}{\pi^*(\alpha^2)} = - \frac{9 + 4 + 6 + 5}{3 + 10 + 9 + 4} = - \frac{2}{4} = 5$$

$$Y_4 = - \frac{Q(\alpha^1)}{\pi^*(\alpha^1)} = - \frac{8 + 1 + 3 + 5}{10 + 8 + 10 + 4} = - \frac{6}{10} = 6$$

b) evaluated by the new formula (24):

$$\bar{A}(x) = 4x^3 + 7x + 6$$

$$\bar{\Pi}(x) = x^4 + 4x^3 + 8x^2 + 8x + 1$$

$$\bar{\Pi}^*(x) = 4x^3 + x^2 + 5x + 8$$

$$n - d = (q-1) - (2t+1) = 3$$

$$Y_1 = - \frac{\bar{A}(\alpha^0)}{\bar{\Pi}^*(\alpha^0)} \alpha^{0 \cdot 3} = \frac{4 + 7 + 6}{4 + 1 + 5 + 8} = \frac{6}{7} = 7$$

$$Y_2 = - \frac{\bar{A}(\alpha^3)}{\bar{\Pi}^*(\alpha^3)} \alpha^{3 \cdot 3} = \frac{2 + 1 + 6}{2 + 9 + 7 + 8} \cdot 6 = \frac{9}{4} \cdot 6 = 3$$

$$Y_3 = - \frac{\bar{A}(\alpha^8)}{\bar{\Pi}^*(\alpha^8)} \alpha^{8 \cdot 3} = \frac{9 + 10 + 6}{9 + 9 + 4 + 8} \cdot 5 = \frac{3}{8} \cdot 5 = 5$$

$$Y_4 = - \frac{\bar{A}(\alpha^9)}{\bar{\Pi}^*(\alpha^9)} \alpha^{9 \cdot 3} = \frac{6 + 9 + 6}{6 + 3 + 8 + 8} \cdot 7 = \frac{10}{3} \cdot 7 = 6$$

The loop in the initialization box of program 16, adapted from Blahut [29], simultaneously computes the erasure locator polynomial and the modified syndrome polynomial for initializing  $b^T(x)$  and  $r^T(x)$ . It is apparent that this loop, or its equivalent, can be added to the initialization box of any of the decoding programs discussed earlier for converting them for the handling of erasures.

Thus, we can appropriately modify any of these programs to yield both  $\pi(x)$  and  $\Omega(x)$  for error-and-erasure decoding of BCH codes. As an example we give program 17, which is an adaptation of program 4 for erasure decoding.

In program 17  $f(x)$  is defined as  $x^{2t}$  and  $g(x)$  as  $S(x)$ , so that at each iteration  $k$  we have the relation

$$|r^k(x)|_{x^{2t}} = |b^k(x)S(x)|_{x^{2t}}$$

holding. In particular, for  $k = -1$ ,  $r^k(x) = x^{2t}$  and  $b^k(x) = 0$  and for  $k = 0$ ,  $r^k(x) = S(x)$  and  $b^k(x) = 1$ . At the conclusion of the initialization loop  $r^k(x) = T(x)$  and  $b^k(x) = \kappa(x)$ . At termination of the program  $r^k(x) = \Omega(x)$  and  $b^k(x) = \pi(x)$ , as desired.

Example 22: Reed-Solomon 3-error-correcting code over  $GF(11)$  with  $\alpha = 2$

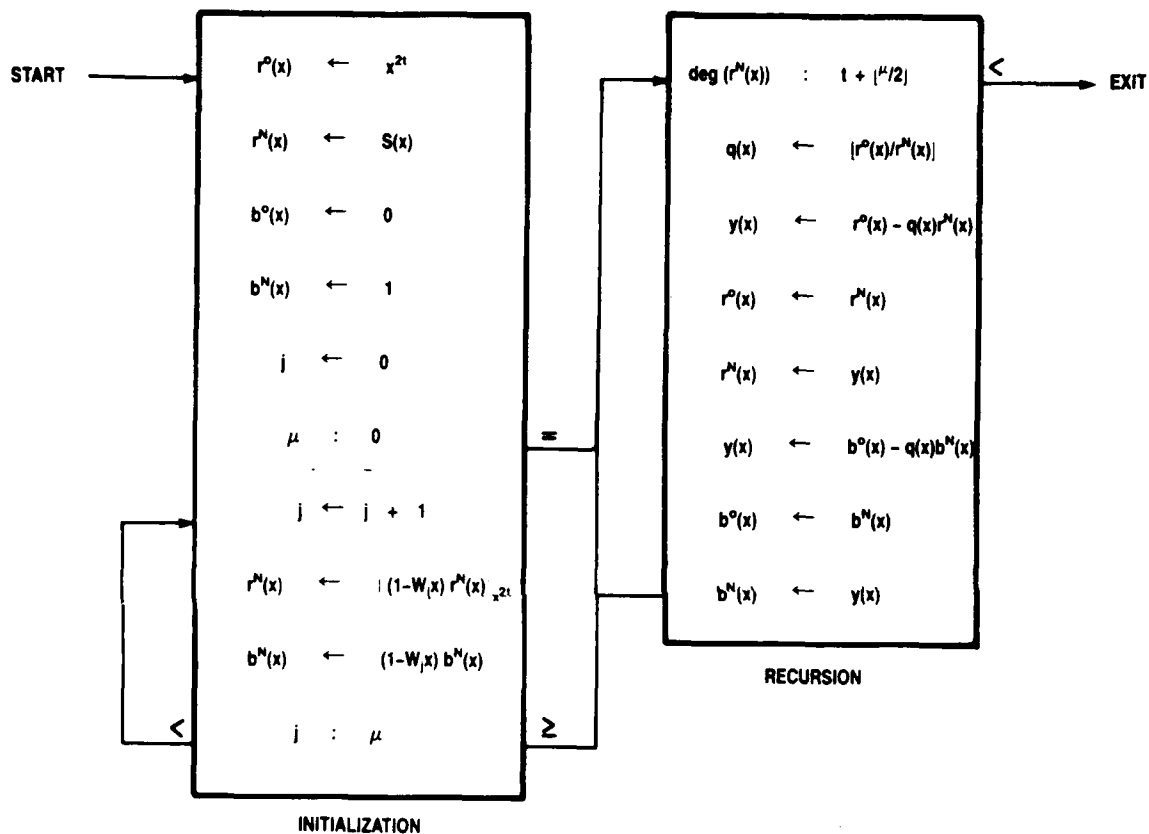
$t = 3$ ; Let  $c(x) = 0$ ,  $e(x) = 3x^3 + 7$ ,  $d(x) = 6x^9 + 5x^8$ .

$$v(x) = c(x) + d(x) + e(x) = 6x^9 + 5x^8 + 3x^3 + 7.$$

$$S(x) = 6x^5 + 3x^4 + 5x^3 + 4x^2 + 9x + 5$$

$$j = 0: r^0(x) = x^{2t}, b^0(x) = 0$$

$$\begin{aligned} r^N(x) &= S(x) = 6x^5 + 3x^4 + 5x^3 + 4x^2 + 9x + 5 \\ b^N(x) &= 1 \end{aligned}$$



INPUT: POLYNOMIALS  $x^{2t}$ ,  $S(x)$ ; ERASURE LOCATIONS  $W_j$ ,  
NUMBER OF ERASURES  $\mu$ , INTEGER  $t$

OUTPUT:  $\gamma\Pi(x) = b^N(x)$ ,  $\gamma\Omega(x) = r^N(x)$

PROGRAM 17: JAPANESE ALGORITHM WITH ERASURES

$$j = 1: W_j = \alpha^8 = 3$$

$$r^N(x) + \left| (1 - 3x)r^N(x) \right|_x 2t = 8x^5 + 10x^4 + 4x^3 + 10x^2 + 5x + 5$$

$$b^N(x) + (1 - 3x)b^N(x) = 8x + 1$$

$$j = 2: W_j = \alpha^9 = 6$$

$$r^N(x) + \left| (1 - 6x)r^N(x) \right|_x 2t = 3x^5 + 8x^4 + 10x^3 + 2x^2 + 8x + 5$$

$$b^N(x) + (1 - 6x)b^N(x) = 7x^2 + 2x + 1$$

$$q(x) + \left\lfloor r^0(x)/r^N(x) \right\rfloor = 4x + 4$$

$$\begin{aligned} r^N(x) + r^0(x) - q(x)r^N(x) &= 5x^4 + 7x^3 + 4x^2 + 3x + 2 \\ b^N(x) + b^0(x) - q(x)b^N(x) &= 5x^3 + 8x^2 + 10x + 7 \end{aligned}$$

$$q(x) + \left\lfloor r^0(x)/r^N(x) \right\rfloor = 5x + 10$$

$$\begin{aligned} r^N(x) + r^0(x) - q(x)r^N(x) &= 8x^3 + 2x^2 + x + 7 \\ b^N(x) + b^0(x) - q(x)b^N(x) &= 8x^4 + 9x^3 + 9x^2 + 10x + 8 \end{aligned}$$

$$\deg(r^N(x)) = 3 < 4 = t + \lfloor \mu/2 \rfloor; \text{ stop.}$$

$$\gamma = 8, \gamma^{-1} = 7$$

$$\Pi(x) = \gamma^{-1}b^N(x) = x^4 + 8x^3 + 8x^2 + 4x + 1$$

$$\Omega(x) = \gamma^{-1}r^N(x) = x^3 + 3x^2 + 7x + 5$$



## SECTION 10

### CONCLUSION

The decoding algorithms of Sugiyama, Kasahara, Hirasawa, and Namekawa, of Mills', and the Berlekamp-Massey algorithm have been reviewed and compared. All can be viewed as variants of Euclid's algorithm. Various enhancements of these algorithms have been considered, including modifications which avoid the computation of finite field inverses, and which permit decoding of erasures in addition to errors.

The Japanese algorithm and Mills' algorithm are based on a direct application of Euclid's algorithm to solve the key equation (14) for BCH decoding. We have seen that when the polynomial divisions contained in these algorithms are broken down into their individual partial divisions the result is a two-loop structure depending on whether a polynomial division is or is not being completed. These decoding algorithms, therefore, appear to be at a disadvantage compared to the single-loop Berlekamp-Massey algorithm.

Treating the Berlekamp-Massey algorithm in a Euclidean context yields the error (or errata) evaluator polynomial in addition to the locator polynomial and obviates the need to perform a vector inner-product calculation for computing the discrepancies. In this form the algorithm appears to be well-suited for VLSI implementation in a systolic array. This implementation will be the subject of further investigation.

# LIST OF REFERENCES

1. Hocquenghem, A., "Codes Correcteurs d'Erreurs," Chiffres 2 (September 1959), pp. 147-156.
2. Bose, R.C. and Ray-Chaudhuri, D.K., "On a Class of Error-Correcting Binary Group Codes," Information and Control 3 (March 1960), pp. 68-79.
3. Bose, R.C. and Ray-Chaudhuri, D.K., "Further Results on Error-Correcting Binary Group Codes," Information and Control 3 (September 1960), pp. 279-290.
4. Peterson, W.W., "Encoding and Error-Correcting Procedures for the Bose-Chaudhuri Codes," IEEE Trans. on Information Theory IT-6 (September 1960), pp. 459-470.
5. Peterson, W.W., Error-Correcting Codes, The MIT Press, Cambridge (1961).
6. Gorenstein, D. and Zierler, N., "A Class of Cyclic Linear Error-Correcting Codes in  $p^m$  Symbols," J. SIAM 9 (June 1961), pp. 207-214.
7. Reed, I.S. and Solomon, G., "Polynomial Codes Over Certain Finite Fields," J. SIAM 8 (June 1960), pp. 300-304.
8. Chien, R.T., "Cyclic Decoding Procedures for Bose-Chaudhuri-Hocquenghem Codes," IEEE Trans. on Information Theory IT-10 (October 1964), pp. 357-363.
9. Forney, G.D., "On Decoding BCH Codes," IEEE Trans. on Information Theory IT-11 (October 1965), pp. 136-144.
10. Berlekamp, E.R., Algebraic Coding Theory, McGraw-Hill, New York (1968).
11. Massey, J.L., "Shift-Register Synthesis and BCH Decoding," IEEE Trans. on Information Theory IT-15 (January 1969), pp. 122-127.

LIST OF REFERENCES (Continued)

12. Sugiyama, Y., Kasahara, M., Hirasawa, S., Namekawa, T., "A Method for Solving Key Equation for Decoding Goppa Codes," Information and Control 27 (January 1975), pp. 87-99.
13. Mills, W.H., "Continued Fractions and Linear Recurrences," Mathematics of Computation 29 (January 1975), pp. 173-180.
14. Welch, L.R., and Scholtz, R.A., "Continued Fractions and Berlekamp's Algorithm," IEEE Trans. on Information Theory IT-25 (January 1979), pp. 19-27.
15. Euclid, Elements, translated by Heath, T.L., Dover, New York (1956).
16. Iverson, K.E., A Programming Language, Wiley, New York (1962).
17. Blahut, R.E., Theory and Practice of Error Control Codes, Addison-Wesley, Reading (1983).
18. Peterson, W.W. and Weldon, E.J., Jr., Error-Correcting Codes, 2d ed., The MIT Press, Cambridge (1967).
19. MacWilliams, F.J., and Sloane, N.J.A., The Theory of Error-Correcting Codes, North-Holland, New York (1977).
20. McEliece, R.J., The Theory of Information and Coding (Volume 3 in The Encyclopedia of Mathematics and its Applications, G.C. Rota, ed.), Addison-Wesley, Reading (1977).
21. Hamming, R.W., Numerical Methods for Scientists and Engineers, McGraw-Hill, New York (1973).

# LIST OF REFERENCES (Continued)

22. Citron, T., "Method and Means for Error Detection and Correction in High Speed Data Transmission Codes," U.S. patent application, Hughes (1985).
23. Kung, S.Y., "Multivariable and Multidimensional Systems: Analysis and Design," Ph.D. Dissertation, Stanford University (1977).
24. Lanczos, C., "An Iteration Method for the Solution of the Eigenvalue Problem of Linear and Integral Operators," J. Res. Nat. Bur. of Standards 45 (1950), pp. 255-282.
25. Schur, J., "Ueber Potenzreihen, die im Innern des Einheitskreises beschaenkt sind," J. Reine Angew. Math. 147, (1917), pp. 205-232.
26. Kailath, T., "Signal Processing in the VLSI Era," in Kung, S.Y., Whitehouse, H.J., Kailath, T., (eds.), VLSI and Modern Signal Processing, Prentice-Hall, Englewood Cliffs (1985), pp. 1-24.
27. Burton, H.O., "Inversionless Decoding of Binary BCH Codes," IEEE Trans. on Information Theory IT-17 (July 1971), pp. 464-466.
28. Shao, H.M., Truong, T.K., Deutsch, L.J., Yuen, J.H., and Reed, I.S., "A VLSI Design of a Pipeline Reed-Solomon Decoder," IEEE Trans. on Computers C-34 (May 1985), pp. 393-403.
29. Blahut, R.E., "Transform Techniques for Error-Control Codes," IBM J. Research and Development 23 (May 1979), pp. 299-315.
30. Sugiyama, Y., Kasahara, M., Hirasawa, S., and Namekawa, T., "An Erasures-and-Errors Decoding Algorithm for Goppa Codes," IEEE Trans. on Information Theory IT-22 (March 1976), pp. 238-241.



# *MISSION of Rome Air Development Center*

*RADC plans and executes research, development, test and selected acquisition programs in support of Command, Control, Communications and Intelligence (C<sup>3</sup>I) activities. Technical and engineering support within areas of competence is provided to ESD Program Offices (POs) and other ESD elements to perform effective acquisition of C<sup>3</sup>I systems. The areas of technical competence include communications, command and control, battle management information processing, surveillance sensors, intelligence data collection and handling, solid state sciences, electromagnetics, and propagation, and electronic reliability/maintainability and compatibility.*